

# R 数据科学（第二版）



# 目录



# 欢迎语

这是“**R 数据科学**”第二版。这本书将告诉你如何用 R 从事数据科学：你将学到如何把数据导入 R，并将其变换为最有用的结构，还有数据转换和可视化。

在这本书中你将找到数据科学的一系列实践技能。正如化学家学习如何清洗试管和配置实验室一样，你将学习如何清洗数据和绘制图表以及其他许多事情，这些技能使数据科学得以进行。在这里你将会找到利用 R 进行这些操作的最佳方法。你将学会如何利用图形的语法、编程文档和可复现研究来节省时间，还将学习如何管理认知资源，以便在清洗、可视化和探索数据时更容易发现新知识。

根据[CC BY-NC-ND 3.0](#)授权，该网站是且将永远是免费的。如果你想要这本书的纸质版，你可以通过[Amazon](#)订购。如果你因为能免费阅读这本书而心存感激，并希望有所回馈，请向[Kākāpō Recovery](#)捐款。*kākāpō*（出现在封面的这种鸟）是新西兰特有的一种濒危鹦鹉，目前仅剩 248 只。

如果你会说其他语言，你可能会对第二版的免费翻译版感兴趣：

- [中文](#)

你可以在<https://mine-cetinkaya-rundel.github.io/r4ds-solutions>找到书中习题的参考答案。

## 欢迎语

请注意，本书采用了贡献者行为准则（Contributor Code of Conduct）。如果您愿意为这本书做出贡献，表示您同意遵守其条款。

## 致谢

本书由 <https://www.netlify.com> 托管，感谢他们对开源软件和社区的支持。

# 序言

欢迎来到“R 数据科学（第二版）”！该版在第一版的基础上做了一些重要修订，删除了我们认为不再有用的内容，增加了原本打算写进第一版的内容，并对文本和代码进行了整体更新，以反映最佳实践中的变化。同时，我们也很高兴迎来一位新的合著者 Mine Çetinkaya-Rundel，他是一位著名的数据科学教育工作者，也是我们在 Posit(前身为 RStudio) 公司的一位同事。

以下是这些重要变化的概述：

- 本书第一部分已更名为“全局游戏”（Whole game）。这部分的目的是在深入细节之前先为读者提供数据科学“全局游戏”的大致情况。
- 本书第二部分称为“可视化”（Visualize）。和第一版相比，这部分对数据可视化工具和最佳实践进行了更为全面的介绍。虽然要获取所有详细信息仍需参考名为 [ggplot2](#) 的书，但第二版已涵盖了很多最重要的可视化技术。
- 本书第三部分称为“转换”（Transform）。这部分增加了关于数字、逻辑向量和缺失值的新章节，这些之前都是数据转换章节的一部分，但现在需要更多的篇幅来覆盖所有细节。
- 本书第四部分称为“导入”（Import）。这是一组新章节，包括读取纯文本文件和电子表格，从数据库中获取数据、处理大数据、将层次结构数据转换为矩形数据，以及从网站抓取数据。

## 序言

- “编程” (Program) 部分保持不变，但已被从头至尾重写，聚焦于函数编写和迭代的最重要部分。函数编写现在包含了如何封装 tidyverse 函数（处理 tidy 评估的挑战）的详细信息，因为这在过去几年中变得更容易、更重要。我们还增加了一个 base 包重要函数的新章节，这些函数你可能在实际遇到的 R 代码中看到。
- “建模” (Modeling) 部分已被删除。我们之前没有足够的篇幅来全面深入地讲解建模，但现在已经有更好的资源可供选择。我们一般推荐使用[tidymodels](#) 包，并推荐阅读由 Max Kuhn 和 Julia Silge 撰写的[Tidy Modeling with R](#)一书。
- “交流” (Communicate) 部分依然保留，但已全面更新。本书第二版利用[Quarto](#)编写，而不是 R Markdown，Quarto 被看作是更具应用前景的工具。

# 引言

数据科学是一门激动人心的学科，它允许你将原始数据转化为理解、洞察力和知识。《R 数据科学》的目标是帮助你学习 R 中最重要的工具，这些工具将使你能够高效且可复现地进行数据科学工作，并在这个过程中享受乐趣。读完这本书后，你将拥有运用 R 最佳部分来应对各种数据科学挑战的工具。

## 你将学到什么

数据科学范围广阔，不可能通过读一本书就完全掌握。本书的目的旨在为你学习 R 这个重要工具提供坚实和足够的基础知识，以便在必要时可以找到进一步学习的资源。典型的数据科学项目包含的步骤大致如图 ?? 所示。

首先，必须将数据导入 (**import**) R 中。这通常意味着你需要从文件、数据库或 Web 应用程序接口 (API) 中获取数据，并将其加载到 R 的一个数据框中。如果无法将数据导入 R，就无法在其上开展数据科学工作！

一旦导入了数据，还需对其进行整齐 (**tidy**)。整齐数据意味着以统一的形式存储数据，使数据集的语义与其存储方式相匹配。简而言之，当数据整齐时，每一列都是一个变量，每一行都是一个观测值。整齐的数据很重要，因为结构一致使你可以将精力集中在回答有关数据的问题上，而不是努力将数据转换成适合不同功能的正确形式。

## 引言

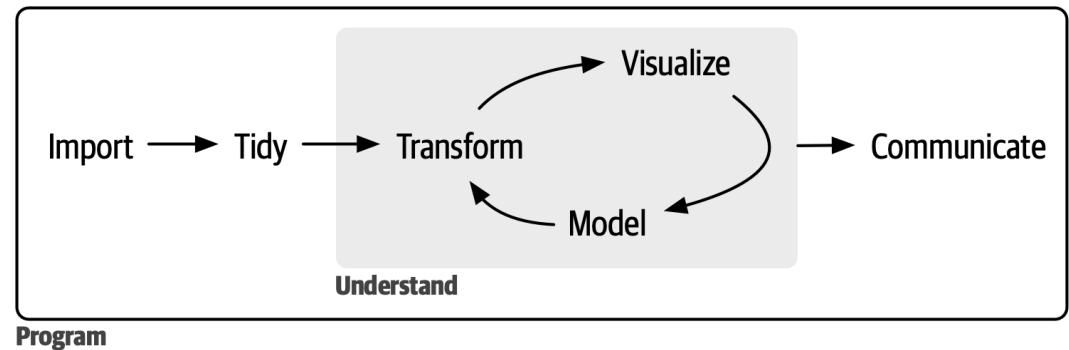


图 1: 在数据科学过程模型中, 首先进行数据导入和整理。接下来, 通过迭代循环的转换、可视化和建模来理解数据。最后, 通过与他人交流结果来完成整个过程。

当数据整齐完成后, 通常下一步是数据转换 (**transform**)。转换包括缩小感兴趣的观察值的范围 (如一个城市的所有人或去年的所有数据)、根据现有变量创建新变量 (如根据距离和时间计算速度), 以及计算一组数据的统计量 (如计数或平均值)。整齐和转换统称为数据清洗 (wrangling) 因为将数据以自然的形式处理通常感觉像是一场战斗!

当有了由所需变量组成的整齐数据后, 就可利用两个主要的知识生成引擎: 可视化和建模。它们的优缺点互补, 因此任何数据分析都可反复利用它们。

可视化 (**Visualization**) 是一项基本的人类活动。一个好的可视化会显示你没有预料到的东西, 或者提出关于数据的新问题。一个好的可视化还可能暗示你问错了问题, 或者你需要收集不同的数据。可视化可能会让你感到惊讶, 但它们的扩展性并不强, 因为它们需要人来解释。

建模 (**Models**) 是对可视化的补充。一旦你的问题足够精准, 你就可以用一个模型来回答。模型基本上是数学或计算工具, 因此它们通常具有良好的可

## 本书的组织结构

扩展性。即使模型扩展性不佳，买更多的电脑通常比买更多的大脑要便宜！但每个模型都需要做假设，而且就其本质而言，模型不能质疑自己的假设。这意味着一个模型不能从根本上给你带来惊喜。

数据科学的最后一步是交流 (**communication**)，这是每个数据分析项目中绝对重要的步骤。除非你能够与他人交流你的结果，否则无论模型和可视化如何帮助你很好地理解数据都无足轻重。

围绕所有这些工具的是编程 (**programming**)。编程是一种综合工具，在数据科学项目的几乎每个部分都要用到。要成为一名成功的数据科学家，你不需要成为一名专业的程序员，但是学习更多的编程知识是值得的，因为成为一名好的程序员可以让你解决常规问题自动化，并使解决新问题变得更轻松。

你将在每个数据科学项目中用到这些工具，但对大多数项目来说这些工具还不够。这里有一个大致的 80/20 规则：你可以使用你在本书中学到的工具来解决项目中大约 80% 的问题，但剩下的 20% 需要其他工具来处理。在本书中我们将为你提供更多了解更多信息的资源。

## 本书的组织结构

前面关于数据科学工具的描述大致上是按照数据分析中使用的顺序组织的（当然，你会多次重复这个过程）。然而，根据我们的经验，首先学习数据导入和整理是次优的，因为 80% 的时间是常规的且无聊的，而另外 20% 的时间是不寻常且令人沮丧的。这不是学习新学科的好起点！相反，我们将从已经导入和整理的数据的可视化和转换开始。这样，当你处理和整理自己的数据时，你的动力就会保持高涨，因为你知道痛苦是值得的。

在每一章中，我们尽量遵循一个一致的模式：从一些鼓舞人心的例子开始，让你可以看到更大的图景，然后再深入到细节中。书中的每一部分都配有练习

## 引言

来帮助你练习所学的知识。虽然跳过练习很有诱惑力，但没有比在实际问题上练习更好的学习方法了。

## 学不到的内容

有几个重要的主题这本书没有涉及。我们认为坚持不懈地专注于最基本的东西是很重要的，这样你就能尽快开始行动。这也表明这本书不可能涵盖每一个重要主题。

## 建模

对于数据科学来说建模是非常重要的，但这是一个很大的主题。而不幸的是，我们没有足够的空间在这里给予它应有的篇幅。要了解更多关于建模的知识，我们强烈推荐由我们的同事 Max Kuhn 和 Julia Silge 编写的 [Tidy Modeling with R](#)。本书将向你介绍 `tidymodels` 包家族，正如你从名称中猜到的那样，它与我们在本书中使用的 `tidyverse` 包共享许多约定。

## 大数据

本书主要关注小型内存数据集。这是一个正确的起点，因为只有在你具备处理小数据的经验后，才能处理大数据。本书大部分内容所提供的工具都可以用来轻松处理数百兆字节的数据，稍加注意它们甚至可以处理几 GB 的数据。我们还将向你展示如何从数据库和 *Parquet* 文件中获取数据，这两种文件通常用于存储大数据。你不一定能够处理整个数据集，但这不是问题，因为你只需要一个子集或者子样本来回答你感兴趣的问题。

## 预备知识

如果你经常需要处理更大的数据（比如说 10-100GB），我们建议你进一步了解 `data.table`。我们在这里不做讲解，因为它使用的接口与 `tidyverse` 不同，需要学习一些不同的约定。然而它的速度非常快，如果你正在处理大数据，那么它的优异性能表现值得你投入一些时间来学习它。

### Python、Julia 和其他编程语言

在这本书中，你不会学到任何关于 Python、Julia 或其他对数据科学有用的编程语言的知识。这并不是因为我们认为这些工具不好，它们很优秀！在实践中，大多数数据科学团队使用混合语言，通常至少是 R 和 Python。但我们坚信，最好一次只掌握一种工具，R 就是一个很好的起点。

### 预备知识

为了让你从这本书中获得最大的收益，我们对你已经知道的内容做了一些假设。你应该对数字有一定的了解，如果你有一些基本的编程经验，那将会很有帮助。如果你以前从未编过程序，你会发现 Garrett 编写的 [Hands on Programming with R](#) 是这本书的一个有益补充。

运行这本书中的代码，你需要四样东西：R、RStudio，一个名为 `tidyverse` 的 R 包集合以及其他几个包。包是可重现 R 代码的基本单元。它们包括可重用的函数、描述如何使用它们的文档以及样本数据。

### R

要下载 R，请访问 CRAN (the **c**omprehensive **R** **a**rchive **n**etwork) <https://cloud.r-project.org>。R 新的主要版本每年发布一次，也会发布 2-3 个次要版

## 引言

本，建议定期更新。升级可能会有点麻烦，特别是主要版本，需要你重新安装所有的包，但是拖延只会让情况变得更糟。我们建议本书使用 R4.2.0 或更高版本。

## RStudio

RStudio 是一个用于编程的集成开发环境，你可以从 <https://posit.co/download/rstudio-desktop/> 下载。

RStudio 每年更新几次，当新版本发布时，它会自动通知你，所以不需要定期查看，但最好定期升级，以利用最新和最强大的功能。对于这本书，请确保你至少有 RStudio 2022.02.0 版本。

当你启动 RStudio 后，如图 ??，在界面中你会看到两个关键区域：控制台面板和输出面板。现在你需要知道的是，你在控制台面板中输入 R 代码，然后按回车键来运行它。随着我们的逐步深入，你会学到更多<sup>1</sup>。

## Tidyverse

你还需要安装一些 R 包。R 包是函数、数据和文档的集合，它扩展了基础 R 的功能。使用包是成功使用 R 的关键。你在本书中学习的大多数包都是所谓 tidyverse 的一部分。tidyverse 中的所有包都共享数据和 R 编程的共同理念，并被精心安排在一起协同工作。

你可以使用一行代码安装完整的 tidyverse 包：

---

<sup>1</sup>If you'd like a comprehensive overview of all of RStudio's features, see the RStudio User Guide at <https://docs.posit.co/ide/user>.

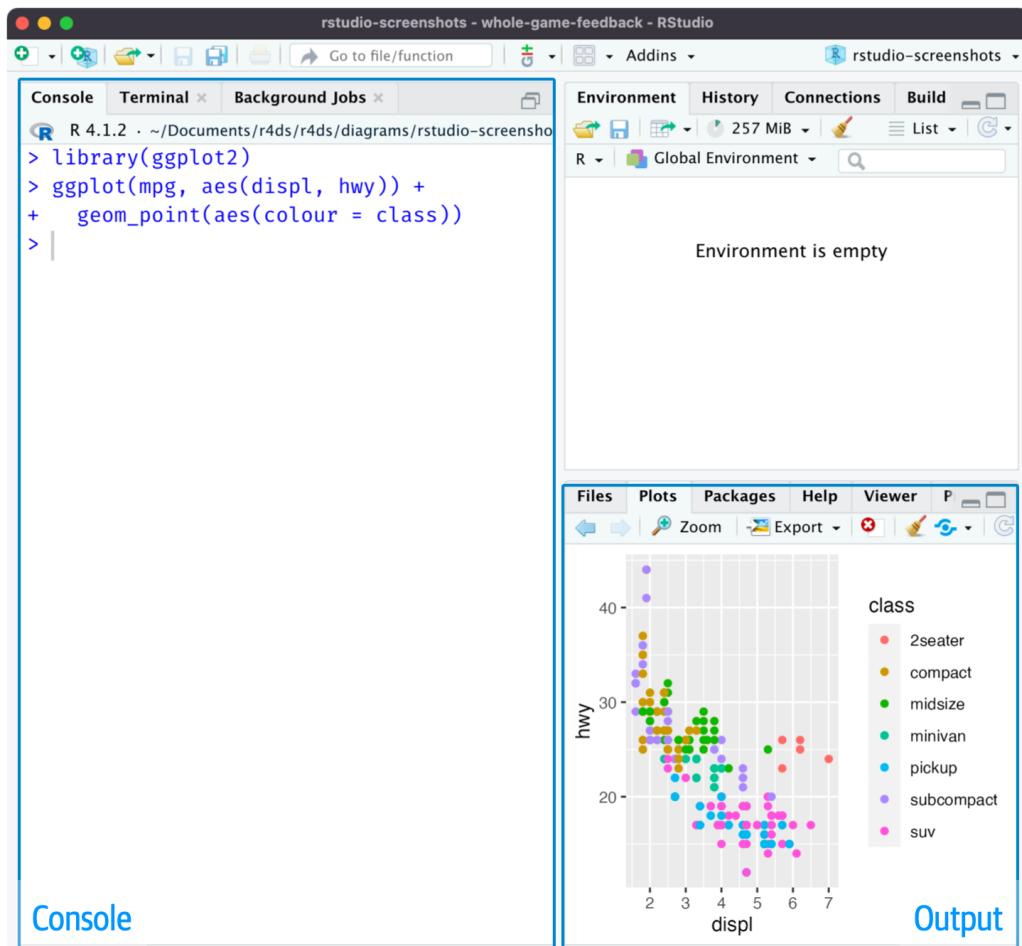


图 2: RStudio IDE 有两个关键区域：在左侧的控制台面板中输入 R 代码，在右侧的输出面板中查看图形。

## 引言

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

在你的计算机上，在控制台中键入这行代码，然后按 enter 键运行它，R 将从 CRAN 下载软件包并安装到你的计算机上。

在使用 `library()` 加载包之前，你无法使用包中的函数、对象或帮助文件。一旦安装了一个包，你可以使用 `library()` 来加载它：

```
library(tidyverse)
#> -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
#> v dplyr     1.1.4     v readr     2.1.5
#> vforcats   1.0.0     v stringr   1.5.1
#> v ggplot2   3.5.0     v tibble    3.2.1
#> v lubridate 1.9.3     v tidyrr    1.3.1
#> v purrr    1.0.2
#> -- Conflicts ----- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()   masks stats::lag()
#> i Use the conflicted package (<a href="http://conflicted.r-lib.org/">http://conflicted.r-lib.org/
```

这告诉你，tidyverse 加载了 9 个包：dplyr、forcats、ggplot2、lubrid、purrr、readr、stringr、tibble 和 tidyrr。它们被认为是 tidyverse 的核心，因为几乎在所有数据分析中都会用到它们。

tidyverse 中的包变化相当频繁。可以通过运行 `tidyverse_update()` 查看是否有可用的更新。

## 其他包

还有许多其他优秀的软件包，它们不是 tidyverse 的一部分，因为它们解决了不同领域的问题，或者是用不同的基本原则设计的。这不会让它们变得更好或更糟，这只会让它们与众不同。换句话说，对 tidyverse 的补充不是 messyverse，而是由相互关联的包组成的许多其他 universes。随着你使用 R 处理更多的数据科学项目，你将学习新的包和新的数据思考方式。

在本书中，我们将使用许多来自 tidyverse 之外的包。例如以下的包为我们学习 R 的过程中提供了有趣的数据集：

```
install.packages(
  c("arrow", "babynames", "curl", "duckdb", "gapminder",
    "ggrepel", "ggridges", "ggthemes", "hexbin", "janitor", "Lahman",
    "leaflet", "maps", "nycflights13", "openxlsx", "palmerpenguins",
    "repurrrsive", "tidymodels", "writexl")
)
```

我们还将使用一些其他包作为单独的示例。你现在不需要安装它们，只要记住每当你看到这样的错误时：

```
library(ggrepel)
#> Error in library(ggrepel) : there is no package called 'ggrepel'
```

你需要运行 `install.packages("ggrepel")` 来安装这个包。

## 引言

### 运行 R 代码

前一节向你展示了几个运行 R 代码的示例，书中的代码看起来是这样的：

```
1 + 2  
#> [1] 3
```

如果你在本地控制台中运行相同的代码，它看起来像这样：

```
> 1 + 2  
[1] 3
```

有两个主要区别：在控制台中，你在 `>` 之后键入，`>` 称为提示符。我们在书中没有显示提示符。在本书中，输出用 `#>` 注释掉。在控制台中，它直接出现在代码之后。这两个区别意味着，如果你使用的是电子书，你可以轻松地从书中复制代码并将其粘贴到控制台。

在本书中，我们使用一致的约定来引用代码：

- 函数用代码字体显示，后面跟着圆括号，如 `sum()` 或 `mean()`；
- 其他 R 对象（例如数据或函数参数）用代码字体，没有圆括号，如 `flights` 或 `x`。
- 有时，为了明确对象来自哪个包，to make it clear which package an object comes from，我们使用包名后面加两个冒号，如 `dplyr::mutate()` 或 `nycflights13::flights`。这也是有效的 R 代码。

## 致谢

### 致谢

这本书不仅仅是 Hadley、我和 Garrett 的作品，也是我们与 R 社区许多人（面对面和在线）多次交谈的结果。非常感谢与你们所有人的交流，非常感谢！

这本书是公开编写的，许多人通过拉取请求（pull requests）做出了贡献。特别感谢通过 GitHub 拉取请求的 259 位贡献者（按用户名字母顺序）：@a-rosenberg, Tim Becker (@a2800276), Abinash Satapathy (@Abinashbunty), Adam Gruer (@adam-gruer), adi pradhan (@adidoit), A.s. (@Adrianzo), Aep Hidayatuloh (@aephidayatuloh), Andrea Gilardi (@agila5), Ajay Deonarine (@ajay-d), @AlanFeder, Daihe Sui (@alan-suidaihe), @alberto-agudo, @AlbertRapp, @aleloji, pete (@alonzi), Alex (@ALShum), Andrew M. (@amacfarland), Andrew Landgraf (@andland), @andyhuynh92, Angela Li (@angela-li), Antti Rask (@AnttiRask), LOU Xun (@aquahead), @ariespirgel, @august-18, Michael Henry (@aviast), Azza Ahmed (@azzaea), Steven Moran (@bambooforest), Brian G. Barkley (@BarkleyBG), Mara Averick (@batpigandme), Oluwafemi OYEDELE (@BB1464), Brent Brewington (@bbrewington), Bill Behrman (@behrman), Ben Herbertson (@benherbertson), Ben Marwick (@benmarwick), Ben Steinberg (@bensteinberg), Benjamin Yeh (@bentyeh), Betul Turkoglu (@betulturkoglu), Brandon Greenwell (@bgreenwell), Bianca Peterson (@BinxiePeterson), Birger Niklas (@BirgerNi), Brett Klamer (@bklamer), @boardtc, Christian (@c-hoh), Caddy (@caddycarine), Camille V Leonard (@camilleleonard), @canovasjm, Cedric Batailler (@cedricbatailler), Christina Wei (@christina-wei), Christian Mongeau (@chrMongeau), Cooper Morris (@coopermor), Colin Gillespie (@csgillespie), Rademeyer Vermaak (@csrvermaak), Chloe Thierstein (@cthierst), Chris Saunders (@ctsa), Abhinav Singh (@curious-abhinav), Curtis Alexander (@curtisalexander),

## 引言

Christian G. Warden (@cwarden), Charlotte Wickham (@cwickham), Kenny Darrell (@darrkj), David Kane (@davidkane9), David (@davidrsch), David Rubinger (@davidrubinger), David Clark (@DDClark), Derwin McGeary (@derwinmcgeary), Daniel Gromer (@dgromer), @Divider85, @djbirke, Danielle Navarro (@djnavarro), Russell Shean (@DOH-RPS1303), Zhuoer Dong (@dongzhuoer), Devin Pastoor (@dpastoor), @DSGeoff, Devarshi Thakkar (@dthakkar09), Julian During (@duju211), Dylan Cashman (@dylancashman), Dirk Eddelbuettel (@eddelbuettel), Edwin Thoen (@EdwinTh), Ahmed El-Gabbas (@elgabbas), Henry Webel (@enryH), Ercan Karadas (@ercan7), Eric Kitaif (@EricKit), Eric Watt (@ericwatt), Erik Erhardt (@erikerhardt), Etienne B. Racine (@etiennebr), Everett Robinson (@evjrob), @fellennert, Flemming Miguel (@flemmingmiguel), Floris Vanderhaeghe (@florisvdh), @funkybluehen, @gabrivera, Garrick Aden-Buie (@gadenbuie), Peter Ganong (@ganong123), Gerome Meyer (@GeroVanMi), Gleb Ebert (@gl-eb), Josh Goldberg (@GoldbergData), bahadir cankardes (@gridgrad), Gustav W Delius (@gustavdelius), Hao Chen (@hao-trivago), Harris McGehee (@harrismcgehee), @hendrikweisser, Hengni Cai (@hengnicai), Iain (@Iain-S), Ian Sealy (@iansealy), Ian Lytle (@ijlyttle), Ivan Krukov (@ivan-krukov), Jacob Kaplan (@jacobkap), Jazz Weisman (@jazzlw), John Blischak (@jdblischak), John D. Storey (@jd-storey), Gregory Jefferis (@jefferis), Jeffrey Stevens (@JeffreyRStevens), 蒋雨蒙 (@JeldorPKU), Jennifer (Jenny) Bryan (@jennybc), Jen Ren (@jenren), Jeroen Janssens (@jeroenjanssens), @jeromecholewa, Janet Wesner (@jilmun), Jim Hester (@jimhester), JJ Chen (@jjchern), Jacek Kolacz (@jkolacz), Joanne Jang (@joannejang), @johannes4998, John Sears (@johnsears), @jonathanflint, Jon Calder (@jonmcalder), Jonathan Page (@jonpage), Jon Harmon (@jonthegeek), JooYoung Seo (@jooyoungseo), Justinas Petuchovas (@jpetuchovas), Jordan (@jrdnbradford), Jeffrey

## 致谢

Arnold (@jrnold), Jose Roberto Ayala Solares (@jroberayalas), Joyce Robbins (@jtr13), @juandering, Julia Stewart Lowndes (@jules32), Sonja (@kaetschap), Kara Woo (@karawoo), Katrin Leinweber (@katrinleinweber), Karandeep Singh (@kdpsingh), Kevin Perese (@kevinxperese), Kevin Ferris (@kferris10), Kirill Sevastyanenko (@kirillseva), Jonathan Kitt (@KittJonathan), @koalabearski, Kirill Müller (@krlmlr), Rafał Kucharski (@kucharsky), Kevin Wright (@kwstat), Noah Landesberg (@landesbergn), Lawrence Wu (@lawwu), @lindbrook, Luke W Johnston (@lwjohnst86), Kara de la Marck (@MarckK), Kunal Marwaha (@marwahaha), Matan Hakim (@matanhakim), Matthias Liew (@MatthiasLiew), Matt Wittbrodt (@MattWittbrodt), Mauro Lepore (@maurolepore), Mark Beveridge (@mbeveridge), @mcewenkhundi, mcsnowface, PhD (@mcsnowface), Matt Herman (@mfherman), Michael Boerman (@michaelboerman), Mitsuo Shioota (@mitsuoxv), Matthew Hendrickson (@mhendrickson), @MJMarshall, Misty Knight-Finley (@mkfin7), Mohammed Hamdy (@mmhamdy), Maxim Nazarov (@mnazarov), Maria Paula Caldas (@mpaulacaldas), Mustafa Ascha (@mustafaascha), Nelson Areal (@nareal), Nate Olson (@nate-dolson), Nathanael (@nateaff), @nattalides, Ned Western (@NedJWestern), Nick Clark (@nickclark1000), @nickelas, Nirmal Patel (@nirmalpatel), Nischal Shrestha (@nischalshrestha), Nicholas Tierney (@njtierney), Jakub Nowosad (@Nowosad), Nick Pullen (@nstjhp), @olivier6088, Olivier Cailloux (@oliviercailloux), Robin Penfold (@p0bs), Pablo E. Garcia (@pabloedug), Paul Adamson (@padamson), Penelope Y (@penelopeysm), Peter Hurford (@peterhurford), Peter Baumgartner (@petzi53), Patrick Kennedy (@pkq), Pooya Taherkhani (@pooyataher), Y. Yu (@PursuitOfDataScience), Radu Grosu (@radugrosu), Ranae Dietzel (@Ranae), Ralph Straumann (@rastrau), Rayna M Harris (@raynamharris), @ReeceGoding, Robin Gertenbach (@rgertenbach), Jajo (@RIngyao), Riva Quiroga (@rivaquiroga), Richard

## 引言

Knight (@RJHKnight), Richard Zijdeman (@rlzijdeman), @robertchu03, Robin Kohrs (@RobinKohrs), Robin (@Robinlovelace), Emily Robinson (@robinsones), Rob Tenorio (@robtenorio), Rod Mazloomi (@RodAli), Rohan Alexander (@RohanAlexander), Romero Moraes (@RomeroBarata), Albert Y. Kim (@rudeboybert), Saghir (@saghirb), Hojjat Salmasian (@salmasian), Jonas (@sauercrowd), Vebash Naidoo (@sciencificity), Seamus McKinsey (@seamus-mckinsey), @seanpwilliams, Luke Smith (@seasmith), Matthew Sedaghatfar (@sedaghatfar), Sebastian Kraus (@sekR4), Sam Firke (@sfirke), Shannon Ellis (@ShanEllis), @shoili, Christian Heinrich (@Shurakai), S'busiso Mkhondwane (@sibuso16), SM Raiyyan (@sm-raiyyan), Jakob Krigovsky (@sonicdoe), Stephan Koenig (@stephan-koenig), Stephen Balogun (@stephenbalogun), Steven M. Mortimer (@StevenM-Mortimer), Stéphane Guillou (@stragu), Sulgi Kim (@sulgik), Sergiusz Bleja (@svenski), Tal Galili (@talgalili), Alec Fisher (@Taurenamo), Todd Gerarden (@tgerarden), Tom Godfrey (@thomasggodfrey), Tim Broderick (@timbroderick), Tim Waterhouse (@timwaterhouse), TJ Mahr (@tjmahr), Thomas Klebel (@tklebel), Tom Prior (@tomjamesprior), Terence Teo (@tteo), @twgardner2, Ulrik Lyngs (@ulyngs), Shinya Uryu (@uribo), Martin Van der Linden (@vanderlindenma), Walter Somerville (@waltersom), @werkstattcodes, Will Beasley (@wibeasley), Yihui Xie (@yihui), Yiming (Paul) Li (@yimingli), @yingxingwu, Hiroaki Yutani (@yutannihilation), Yu Yu Aung (@yuyu-aung), Zach Bogart (@zachbogart), @zeal626, Zeki Akyol (@zekiakyol).

## 版权

本书的在线版本访问 <https://r4ds.hadley.nz>，在纸质书印刷期间本书将继续发展。本书的源代码可在 <https://github.com/hadley/r4ds> 获取。本书由 Quarto 提供支持，这使得编写结合了文本和可执行代码的书变得很容易。



## **Part I**

### **全貌概览**



本书这一部分的目的是让你快速浏览数据科学的主要工具：导入、整理、转换和可视化数据，如图 ?? 所示。我们将向你展示数据科学的“全貌”，为你提供各主要部分的足够内容，以便你能够处理真实（尽管可能简单）的数据集。本书后面的部分将深入探讨这些主题，从而增加你解决数据科学挑战的范围。

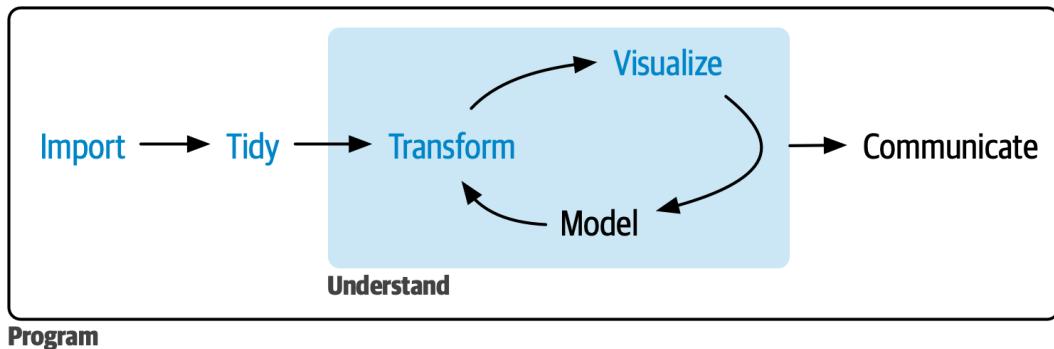


图 3: 在本书的这一节中，你将学习如何导入、整齐、转换和可视化数据；编程贯穿于各个过程。

数据科学工具包含四章内容：

- 可视化是学习 R 编程非常好的起点，因为它的回报是如此明显：你可以制作优雅且信息丰富的图形，以帮助你理解数据。在章节 ?? 中，你将深入了解可视化，学习 ggplot2 图形的基本结构，以及将数据转换为图形的强大技术。
- 仅仅可视化通常是不够的，因此在章节 ?? 中，你将学习关键函数，这些关键函数允许你选择重要变量、筛选关键观测值、创建新变量和计算汇总统计量。
- 在章节 ?? 中，你将学习整理数据，这是一种致的数据存储方式，可以使数据转换、可视化和建模更容易。你将了解其基本原理，以及如何将数据

整理成整齐的形式。

- 在转换和可视化数据之前，首先需要将数据导入 R。在章节 ?? 中将学习把.csv 文件导入 R 的基础知识。

在这些章节中，还有另外四章是关于 R 工作流程的。在章节 ??，章节 ?? 和章节 ?? 中，你将学习编写和组织 R 代码的良好工作流程。从长远来看这会让你更容易取得成功，因为它们将为你提供在处理实际项目时保持条理清晰的工具。最后，章节 ?? 将教你如何获得帮助并继续学习。

# 1 数据可视化

## 1.1 引言

“简单的图表比任何其他工具都能给数据分析师带来更多的信息。”

— John Tukey

R 虽然有几个制图系统，但 **ggplot2** 是最优雅、功能最多的一个系统。**ggplot2** 实现了描述和构建图形的连贯系统，即图形的语法。基于 **ggplot2**，通过学习一个系统并将其应用于许多地方，你可以做更多的事情而且速度更快。

本章将告诉你如何利用 **ggplot2** 可视化数据。我们将首先创建一个简单的散点图，并用它引入 **ggplot2** 的基本构建模块，美学映射和几何对象。然后将引导你可视化单个变量的分布以及可视化两个或多个变量之间的关系。最后，保存所做的图和一些故障排除提示。

### 1.1.1 必要条件

本章重点介绍 **ggplot2**，它是 **tidyverse** 的核心包之一。要访问本章中使用的数据集、帮助页和函数，请运行以下命令加载 **tidyverse**：

## 1 数据可视化

```
library(tidyverse)
#> -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
#> v dplyr     1.1.4     v readr     2.1.5
#> v forcats   1.0.0     v stringr   1.5.1
#> v ggplot2   3.5.0     v tibble    3.2.1
#> v lubridate 1.9.3     v tidyrr    1.3.1
#> v purrr    1.0.2
#> -- Conflicts -----
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()   masks stats::lag()
#> i Use the conflicted package (<a href="https://conflicted.r-lib.org/">https://conflicted.r-lib.org/
```

这一行代码将加载 tidyverse，这是在几乎所有数据分析中都会用到的包。它还告诉您 tidyverse 中的哪些函数与 base R 中的函数冲突（或者与您可能加载的其他包冲突）<sup>1</sup>。

如果运行这段代码得到了错误信息，`there is no package called 'tidyverse'`，你需要先安装它，然后再次运行 `library()`。

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

每个包只需要安装一次，但每次启动新会话时都需要加载它。

除了 tidyverse，我们还会用到 **palmerpenguins** 包，其中包含 **penguins** 数据集，记录了 Palmer 群岛三个岛屿上企鹅的身体测量数据，以及 **ggthemes** 包，它提供了一个色盲安全调色板。

---

<sup>1</sup>你可以通过使用冲突包（conflicted package）来消除该消息，并按需强制解决冲突，这在你加载更多包时变得尤为重要。可以在<https://conflicted.r-lib.org/>上了解更多关于此的信息。

```
library(palmerpenguins)
library(ggthemes)
```

## 1.2 第一步

长鳍企鹅的体重是比短鳍企鹅的体重大还是小？你可能已经有了答案，但试着让你的答案更精确。鳍长和体重之间的关系是什么样的？正相关还是负相关？线性的还是非线性的？这种关系会因企鹅的种类而异吗？企鹅生活的岛屿不同会有影响吗？让我们创建可视化图形来回答这些问题。

### 1.2.1 penguins 数据框

您可以使用包 `palmerpenguins`(`palmerpenguins::penguins`)中的 `penguins` 数据框架来测试你对这些问题的答案。数据框架是变量(列)和观测(行)的矩形集合。`penguins` 收录了 344 条观测，由 Kristen Gorman 博士和南极科考站帕尔默站收集并提供<sup>2</sup>。

为了使讨论更容易，让我们定义一些术语：

- **变量**: 变量是可以测量的数量、质量或属性。
- **值**: 值是你测量一个变量时的状态，一个变量的值可能会随着测量的不同而变化。

---

<sup>2</sup>Horst AM, Hill AP, Gorman KB (2020). `palmerpenguins`: Palmer Archipelago (Antarctica) penguin data. R package version 0.1.0. <https://allisonhorst.github.io/palmerpenguins/>. doi: 10.5281/zenodo.3960218.

## 1 数据可视化

- **观测**: 观测是在相似条件下进行的一组测量（通常在相同的时间和相同的对象上进行）。一个观测会包含几个值，每个值与不同的变量相关联。我们有时将一个观测称为一个数据点。
- **表格数据**: 表格数据是一组值，每个值与一个变量和一个观测相关。如果将每个值放在自己的“单元格”中，将每个变量放在自己的列中，将每个观测放在自己的行中，则表格数据是整洁的。

在这种情况下，变量指的是所有企鹅的一个属性，而观测值指的是单个企鹅的所有属性。

在控制台中键入数据框的名称，R 会输出其内容的预览。需要注意的是这个预览的顶部显示有 **tibble**。在 tidyverse 中，我们使用称为 **tibbles** 的特殊数据框，随后你将了解到更多。

```
penguins
#> # A tibble: 344 x 8
#>   species     island   bill_length_mm bill_depth_mm flipper_length_mm
#>   <fct>      <fct>           <dbl>          <dbl>            <int>
#> 1 Adelie    Torgersen       39.1          18.7            181
#> 2 Adelie    Torgersen       39.5          17.4            186
#> 3 Adelie    Torgersen       40.3           18             195
#> 4 Adelie    Torgersen        NA             NA              NA
#> 5 Adelie    Torgersen       36.7          19.3            193
#> 6 Adelie    Torgersen       39.3          20.6            190
#> # i 338 more rows
#> # i 3 more variables: body_mass_g <int>, sex <fct>, year <int>
```

这个数据框包含 8 列。如果使用 `glimpse()`，你可以看到所有变量和每个

变量的前几个观察值。如果你在使用 RStudio，运行 `View(penguins)` 会打开一个交互式数据预览器。

```
glimpse(penguins)
#> Rows: 344
#> Columns: 8
#> $ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A~
#> $ island        <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge~
#> $ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.~
#> $ bill_depth_mm   <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.~
#> $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, ~
#> $ body_mass_g     <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347~
#> $ sex             <fct> male, female, female, NA, female, male, female, m~
#> $ year            <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2~
```

`penguins` 中的变量包括：

1. `species`: 企鹅的种类 (Adelie, Chinstrap, or Gentoo)
2. `flipper_length_mm`: 企鹅鳍的长度, 以毫米为单位
3. `body_mass_g`: 企鹅的体重, 以克为单位。

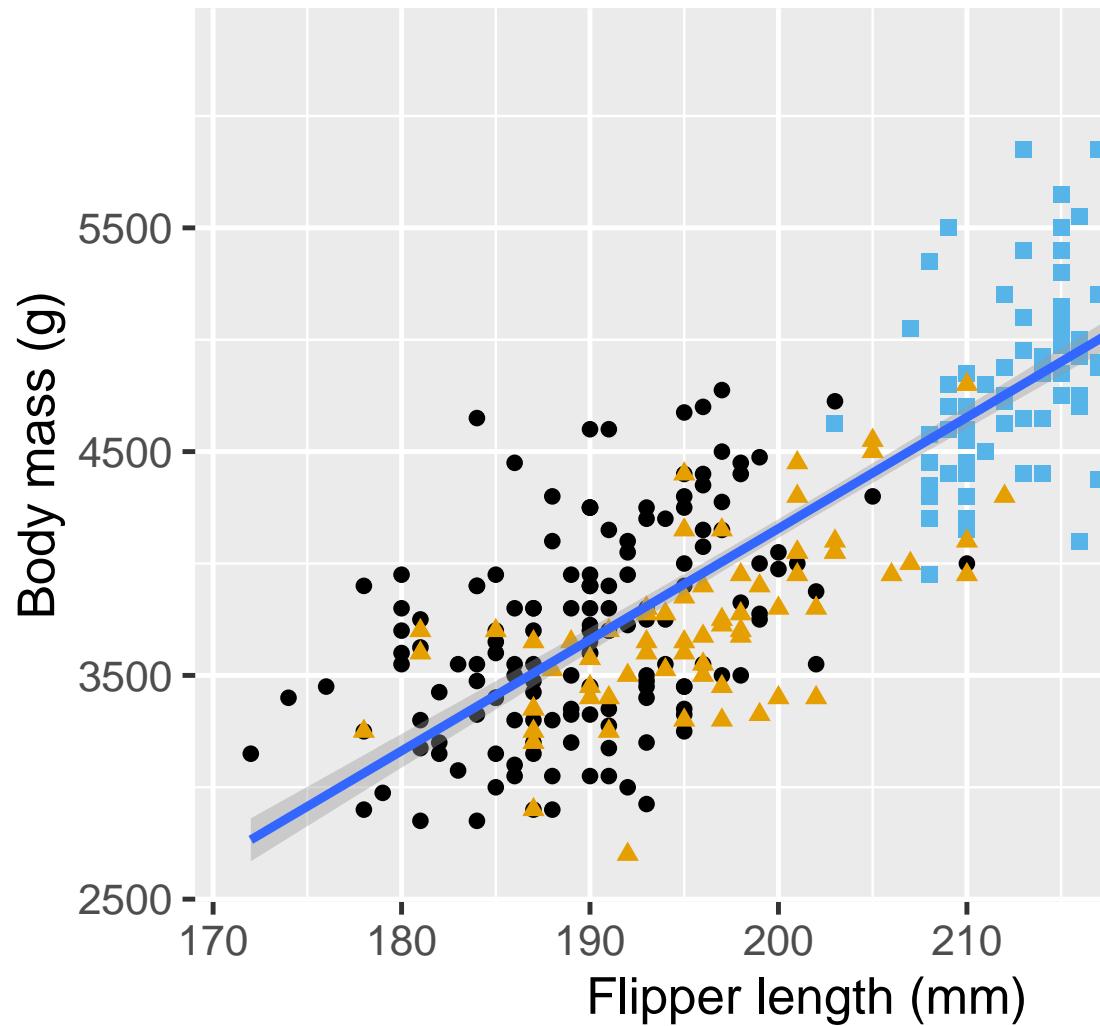
要了解更多关于 `penguins` 的信息, 可以运行`?penguins` 打开帮助页面。

### 1.2.2 最终目标

本章的最终目标是在考虑企鹅种类的情况下重建展示企鹅鳍长和体重之间关系的视图。

## Body mass and flipper length

### Dimensions for Adelie, Chinstrap, and G



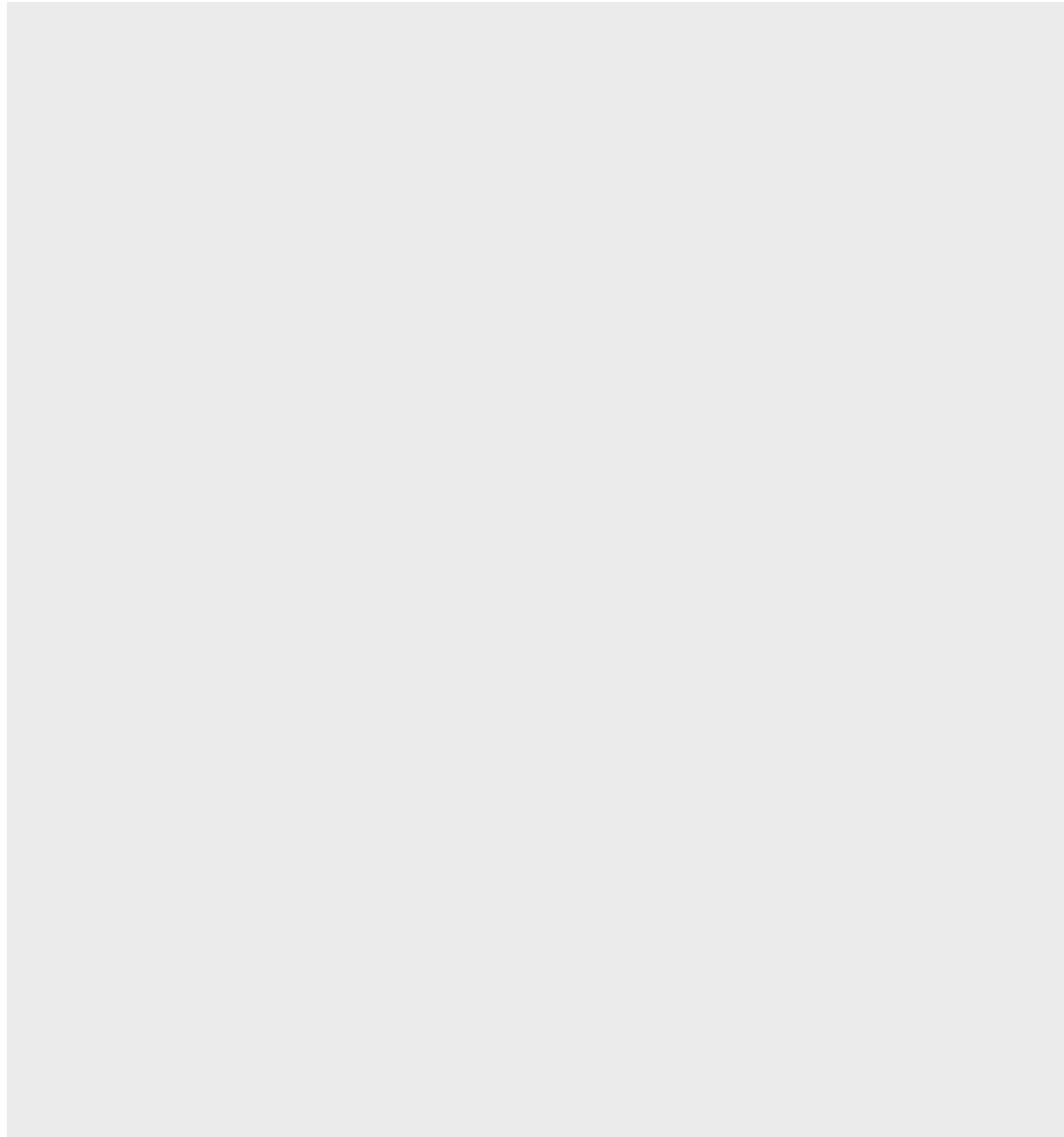
### 1.2.3 构建 `ggplot`

我们一步一步重建这个图。

利用 ggplot2 包的函数 `ggplot()` 绘图。首先定义一个绘图对象，然后向其添加图层。`ggplot()` 的第一个参数是在图中使用的数据集，因此 `ggplot(data = penguins)` 创建了一个空图，用于显示 `penguins` 数据，但由于我们还没有告诉它如何可视化它，所以现在它是空的。这不是一个可以让人动心的图，但你可以把它想象成一块空白画布，你可以在上面画出剩下的图层。

```
ggplot(data = penguins)
```

## 1 数据可视化



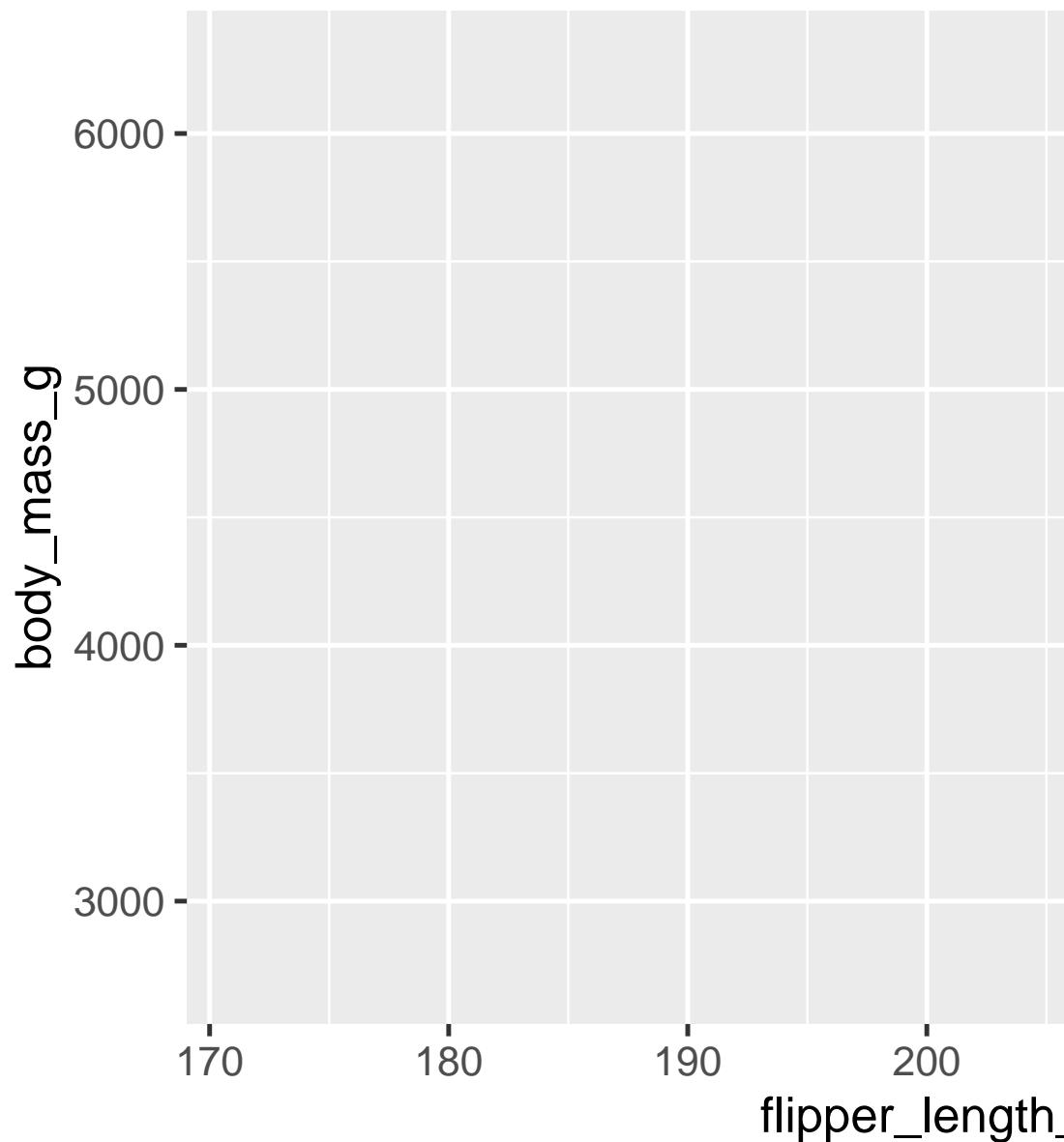
## 1.2 第一步

接下来，需要告诉 `ggplot()` 如何将数据中的信息以视觉方式呈现。`ggplot()` 函数的 `mapping` 参数定义了数据集中的变量如何映射到图表的视觉属性（`aesthetics`）。`mapping` 参数总是在 `aes()` 函数中定义，而 `aes()` 的 `x` 和 `y` 参数则指定了哪些变量映射到 `x` 轴和 `y` 轴上。现在，我们只将鳍长映射到 `x` 轴的美学属性上，将体重映射到 `y` 轴的美学属性上。`ggplot2` 会在数据参数中查找映射的变量，在这个例子中就是 `penguins`。

下图显示了添加这些映射的结果。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g)  
)
```

## 1 数据可视化



## 1.2 第一步

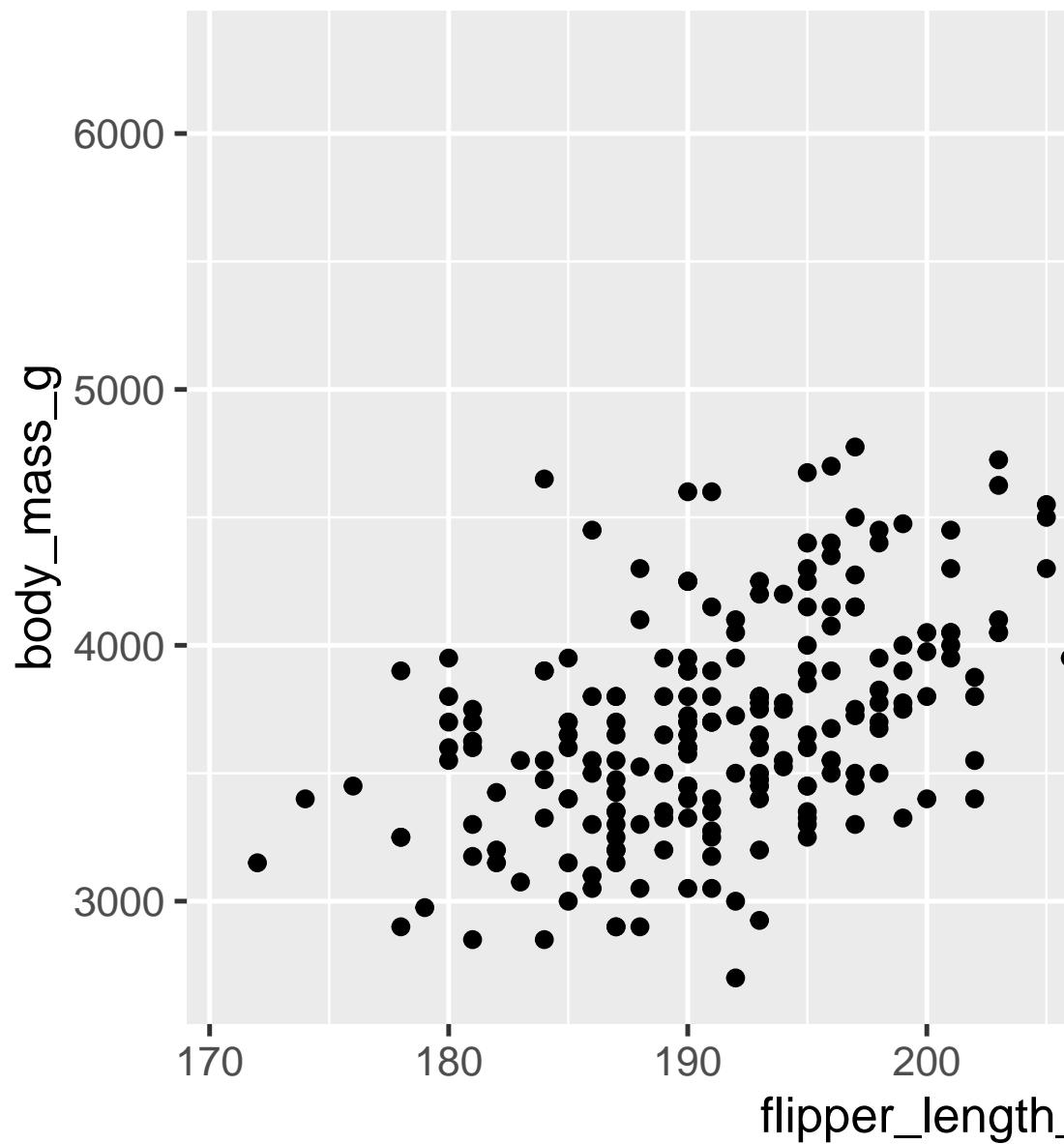
空画布现在有了更多构造，我们已清楚知道鳍的长度在哪里显示（x 轴），身体质量在哪里显示（y 轴）。但 `penguins` 本身还没有出现在图中。这是因为在代码中，我们还没有明确说明如何在图上表示来自数据框架的观测。

为此我们需要定义一个 `geom`（几何对象），`geom` 是一个用来表示数据的图形。这些 `geom` 在 `ggplot2` 中使用以 `geom_` 开头的函数。我们通常利用图使用的 `geom` 类型来描绘图形。例如，条形图使用 `geom_bar()`，折线图使用 `geom_line()`，箱形图使用 `geom_boxplot()`，散点图使用 `geom_point()`，等等。

函数 `geom_point()` 为图添加了一个点图层，从而创建了散点图。`ggplot2` 附带了许多 `geom` 函数，每个函数都为绘图添加不同类型的层。你将在书中学到很多 `geom`，特别是在章节 `??` 中。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g)  
) +  
  geom_point()  
#> Warning: Removed 2 rows containing missing values or values outside the scale range  
#> (`geom_point()`).
```

## 1 数据可视化



## 1.2 第一步

现在我们得到了一些我们可能认为是“散点图”的东西，但这还不是我们的“最终目标”。但利用这个图可以开始回答当初激发我们探索兴趣的问题：“鳍长和体重之间的关系是什么？”这种关系似乎是正的（随着鳍的长度增加体重也会增加），呈线性（这些点聚集在一条直线上，而不是一条曲线），且线性关系比较强（在这样一条直线周围没有太多的发散点）。就体重而言，脚蹼较长的企鹅通常体型较大。

在添加更多图层到这个图之前，让我们暂停一下，回顾一下我们得到的警告信息：

```
Removed 2 rows containing missing values (geom_point()).
```

我们看到这条信息是因为我们的数据集中有两只企鹅缺少体重和（或）鳍长值，如果没有这两个值，ggplot2 无法在图上表示它们。像 R 一样，ggplot2 也秉承这样一种理念：缺失值永远不应该无声地消失。这种类型的警告可能是你在处理实际数据时最常见的警告类型之一。缺失值是一个非常常见的问题，你将在本书中了解更多关于它们的信息，特别是在章节 ?? 中。对于本章剩下的部分，我们将忽略这个警告，这样它就不会出现在我们制作的每个图的旁边。

### 1.2.4 添加美学和图层

散点图对于展示两个数值变量之间的关系非常有用，但是对于两个变量之间任何明显的关系都应持怀疑态度，并询问是否有其他变量可以解释或改变这种明显关系的性质，这总是一个好的思路。例如，不同物种的鳍长和体重的关系是否有所不同？让我们将物种纳入图中，看看这是否能揭示出这些变量之间明显关系的任何额外见解。我们通过使用不同颜色的点代表不同物种来实现这一点。

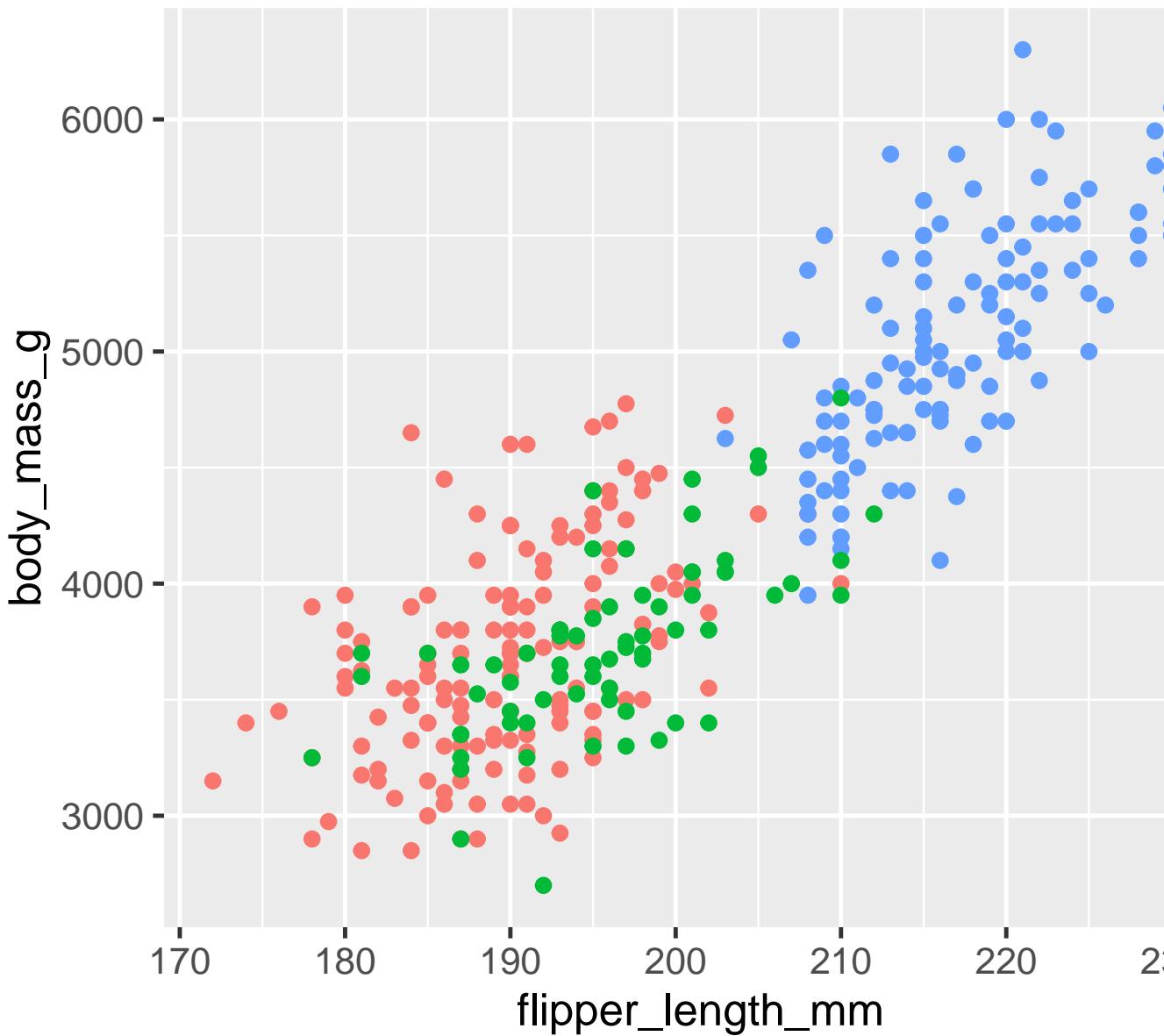
为了实现这一点，我们需要修改美学还是 geom？如果你猜测是函数 `aes()` 内的美学映射，那么表明你已经掌握了使用 ggplot2 进行数据可视化的方法！如

## 1 数据可视化

果不是这样也不用担心，在本书中你将制作更多的 ggplot 图，在制图的过程中有很多机会验证你的直觉。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)  
) +  
  geom_point()
```

## 1.2 第一步



## 1 数据可视化

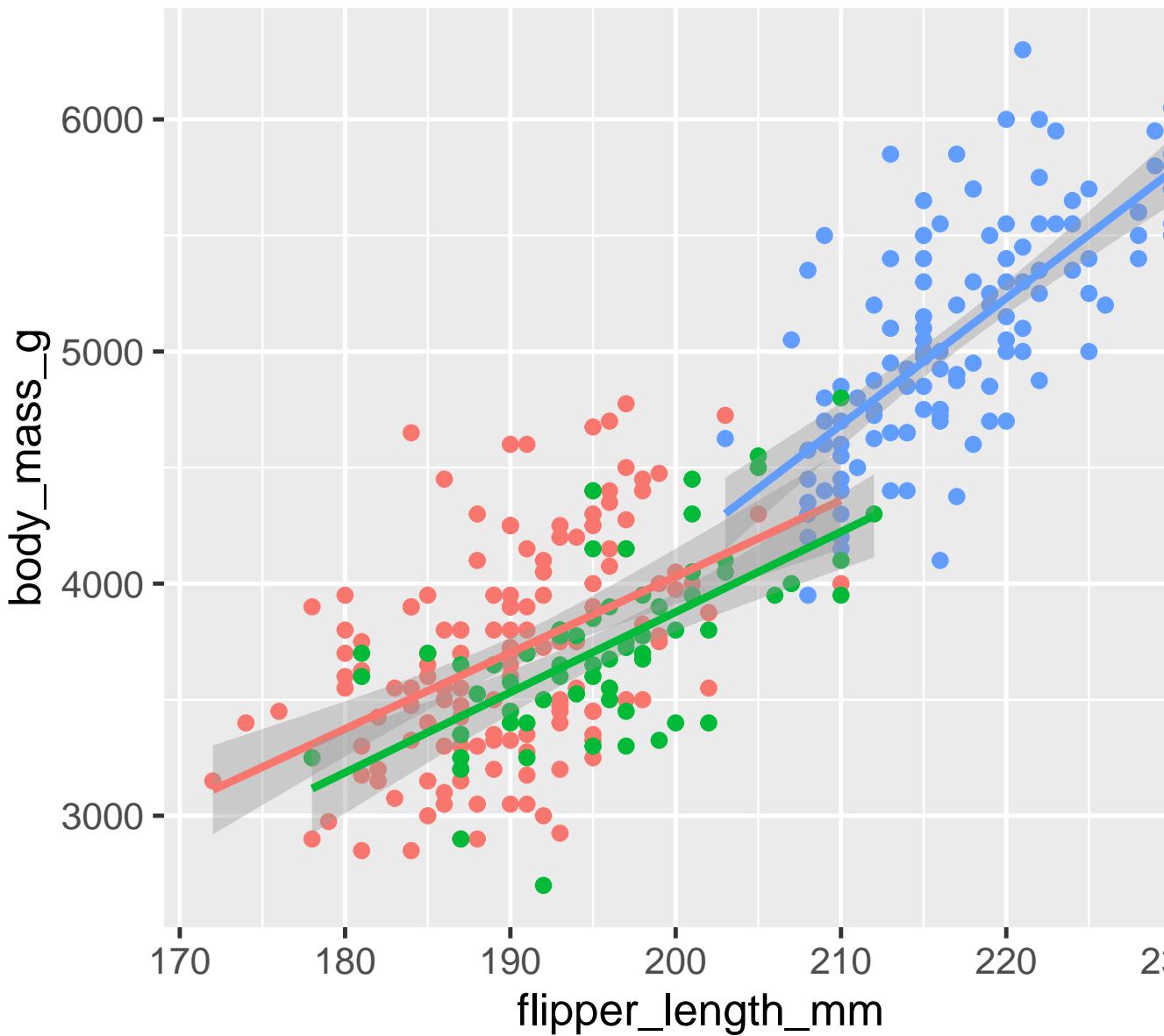
当分类变量被映射到美学上时，ggplot2 将自动为变量的每个水平（三个物种中的一个）分配一个唯一的美学值（这里是某种颜色），这个过程被称为 **scaling**。ggplot2 还将添加一个图例，解释哪些值对应于哪些水平。

现在让我们再添加一层：一条显示体重和鳍长之间关系的平滑曲线。在继续之前请参考上面的代码，并考虑如何将其添加到现有的图中。

由于这是一个代表数据的新 geom，我们将在我们的点几何上添加一个新的 geom 作为图层：`geom_smooth()`。我们将指定基于 `method = "lm"` 的线性模型（linear model）绘制最佳拟合直线。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)  
) +  
  geom_point() +  
  geom_smooth(method = "lm")
```

## 1.2 第一步



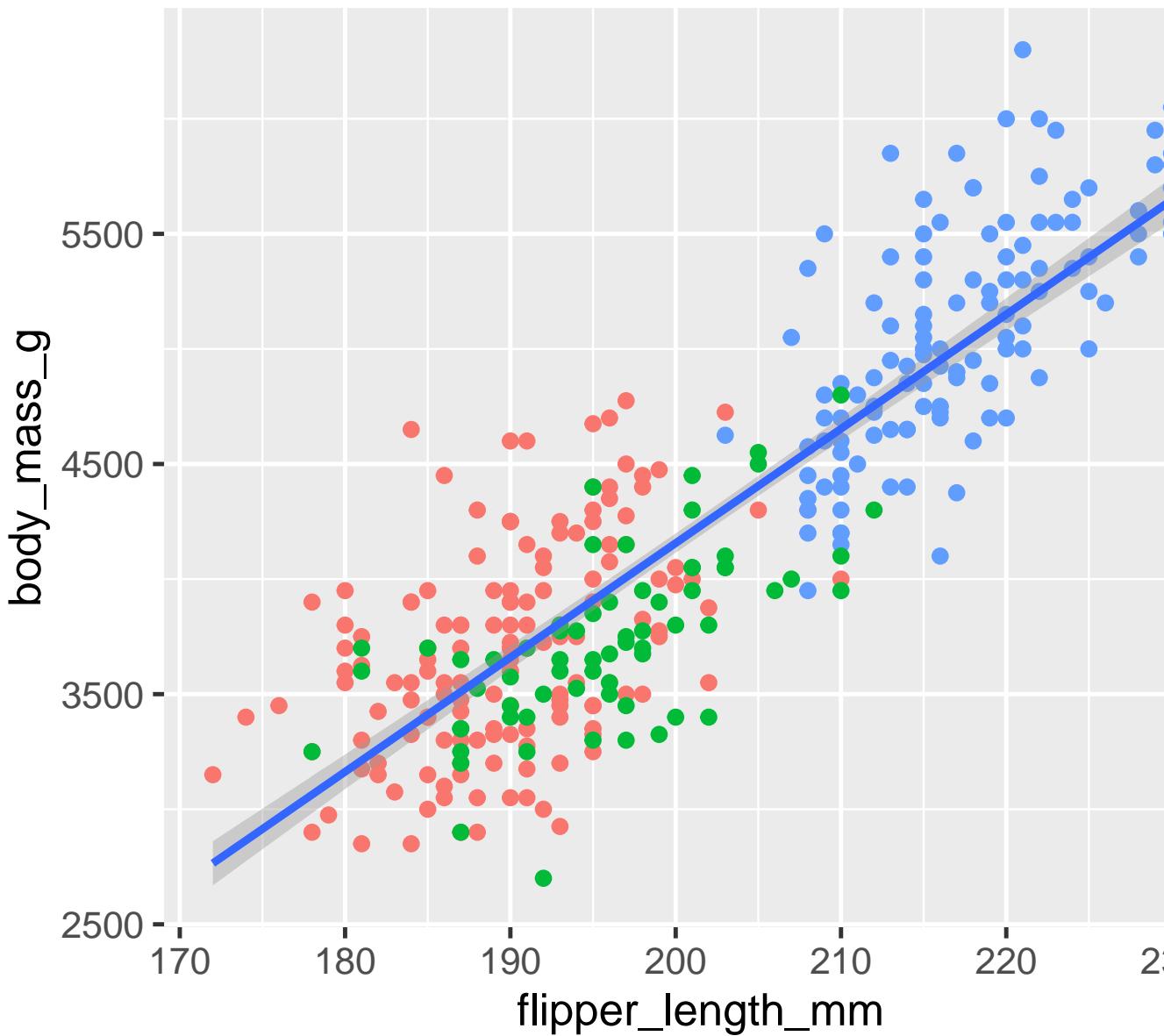
## 1 数据可视化

我们已经成功地添加了直线，但是这个图和第 2 小节 ?? 节中的图看起来并不像，那里的图形整个数据集只有一条直线，而这里每个物种都有单独的一条直线。

在 `ggplot()` 中定义的美学映射 (aesthetic mappings) 在全局级别时，会被传递到图的每一个后续的几何层 (geom layers)。然而，`ggplot2` 中的每个 `geom` 函数也可以接受一个映射参数 (mapping argument)，这允许在局部级别定义美学映射，这些映射会添加到从全局级别继承的映射中。由于我们希望点 (points) 的颜色基于物种 (species) 来设置，但不想为它们将线 (lines) 分开，我们应该只在 `geom_point()` 中指定 `color = species`。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g)  
) +  
  geom_point(mapping = aes(color = species)) +  
  geom_smooth(method = "lm")
```

## 1.2 第一步



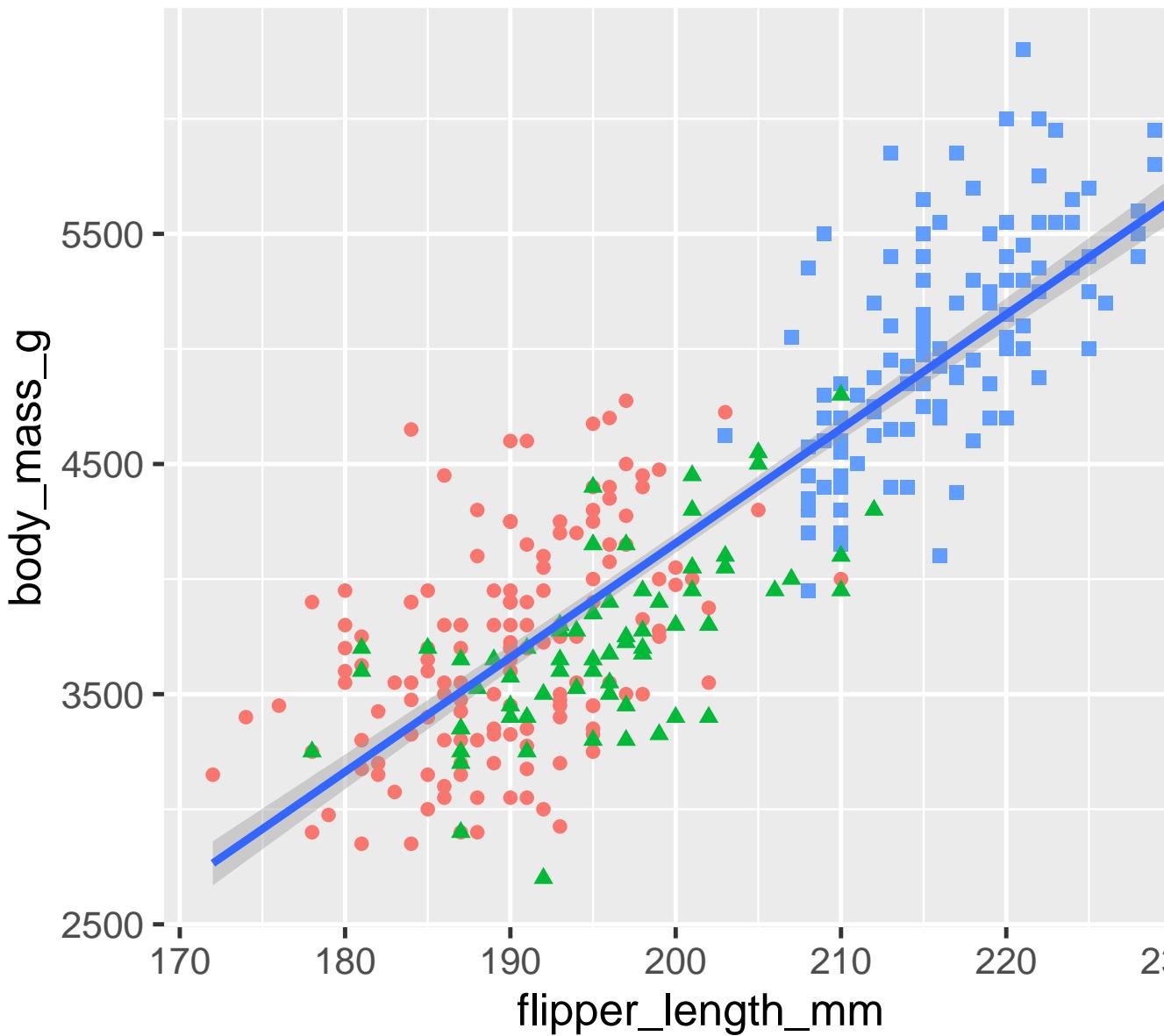
## 1 数据可视化

瞧！我们有了一些看起来很像我们最终目标的东西，尽管它还不完美。我们仍然需要为每种企鹅使用不同的形状，并改进标签。

在图表上仅使用颜色来表示信息通常不是一个好主意，因为由于色盲或其他色觉差异，人们对颜色的感知会有所不同。因此，除了颜色之外，我们还可以将物种映射到形状美学上。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g)  
) +  
  geom_point(mapping = aes(color = species, shape = species)) +  
  geom_smooth(method = "lm")
```

## 1.2 第一步



## 1 数据可视化

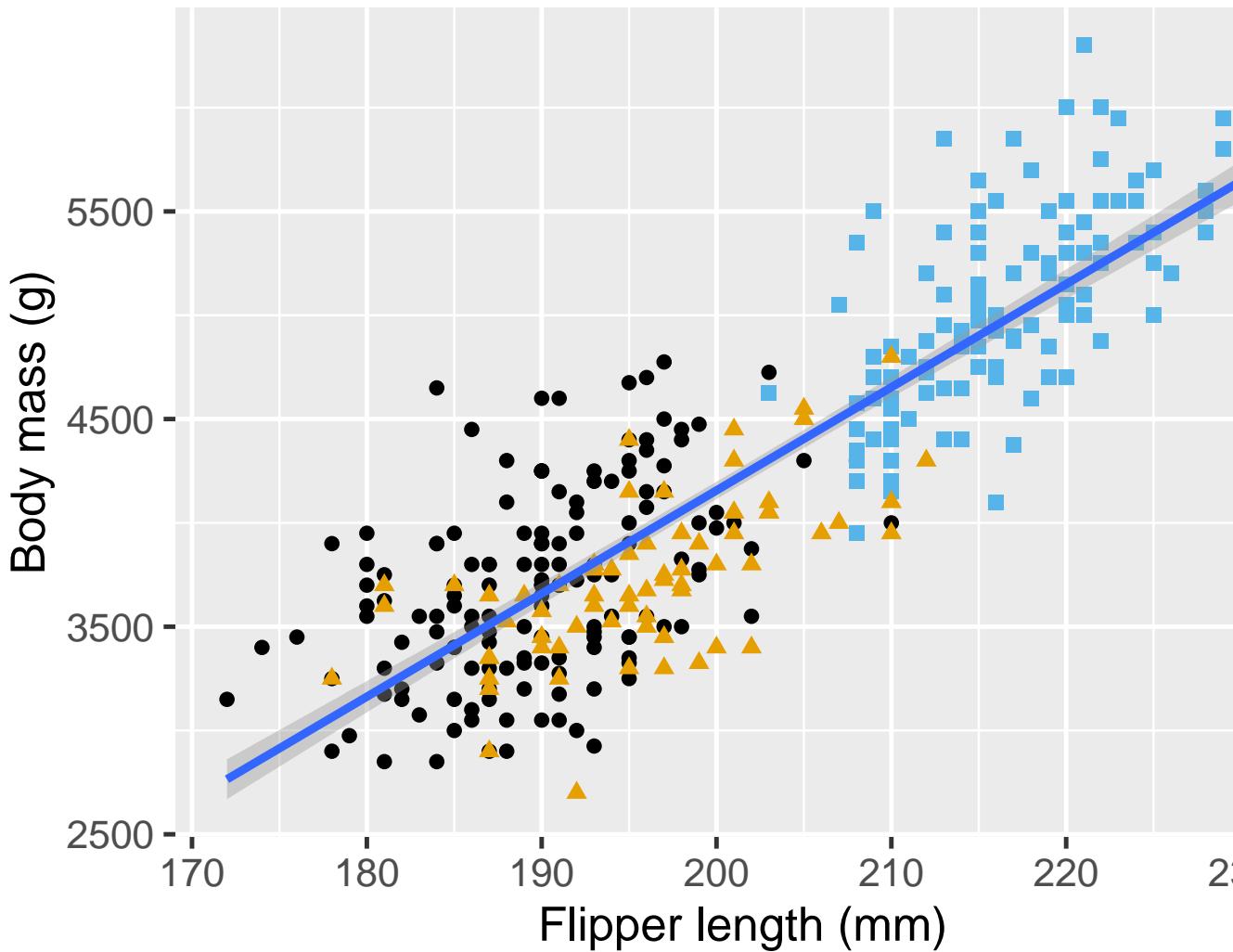
注意，图例也会自动更新以反映点的不同形状。

最后，我们使用 `labs()` 函数在一个新的层中改进图的标签。`labs()` 函数的一些参数含义是不言自明的：`title` 用于添加标题，`subtitle` 用于添加副标题。其他参数与美学映射相对应，`x` 是 x 轴的标签，`y` 是 y 轴的标签，而 `color` 和 `shape` 则定义了图例中的标签。此外，我们可以使用来自 `ggthemes` 包的 `scale_color_colorblind()` 函数来改进颜色调色板，以确保它对色盲用户也是安全的。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g)  
) +  
  geom_point(aes(color = species, shape = species)) +  
  geom_smooth(method = "lm") +  
  labs(  
    title = "Body mass and flipper length",  
    subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",  
    x = "Flipper length (mm)", y = "Body mass (g)",  
    color = "Species", shape = "Species"  
) +  
  scale_color_colorblind()
```

## Body mass and flipper length

Dimensions for Adelie, Chinstrap, and Gentoo Pengu



## 1 数据可视化

我们终于有了一个完全符合我们“最终目标”的图形!

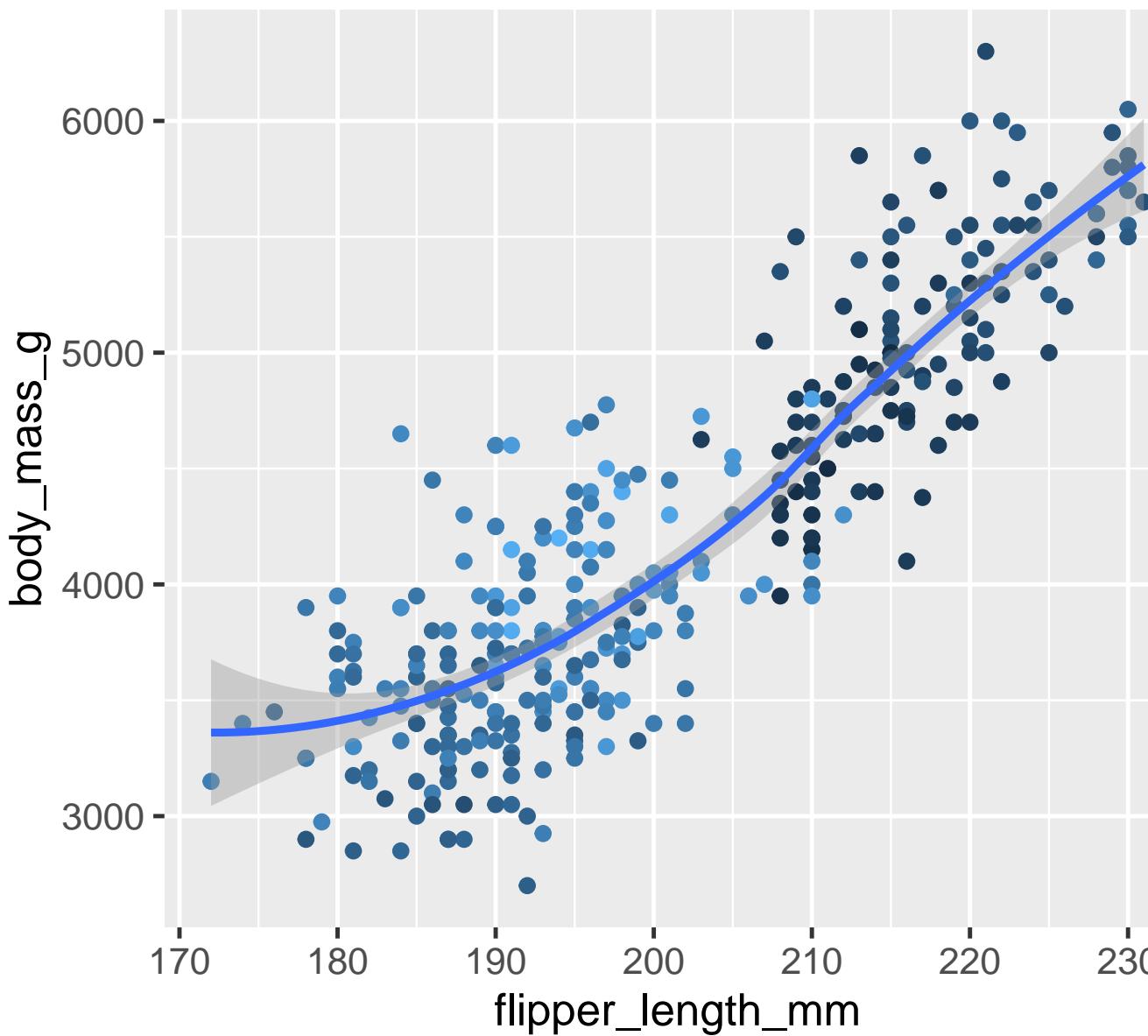
### 1.2.5 练习

1. `penguins` 有多少行? 多少列?
2. 数据框 `penguins` 中的 `bill_depth_mm` 变量是什么? 阅读 `?penguins` 的帮助文件后找到答案;
3. 制作 `bill_depth_mm` 和 `bill_length_mm` 的散点图; 也就是说, 创建一个散点图, y 轴为 `bill_depth_mm`, x 轴为 `bill_length_mm`。描述这两个变量之间的关系;
4. 如果你做一个 `species` 与 `bill_depth_mm` 的散点图会发生什么? 什么几何对象是更好的选择?
5. 为什么下面的代码会给出一个错误, 如何修复它?

```
ggplot(data = penguins) +  
  geom_point()
```

6. `na.rm` 参数在 `geom_point()` 中起什么作用? 这个参数的默认值是什么? 创建一个散点图, 并在其中将这个参数设置为 `TRUE`。
7. 在你之前练习的图中添加以下标题: “数据来自 `palmerpenguins` 包。” 提示: 查看 `labs()` 函数的文档。
8. 重新创建以下可视化。`bill_depth_mm` 应该映射到哪个美学属性上? 它应该在全局级别映射还是在几何对象级别映射?

## 1.2 第一步



## 1 数据可视化

9. 在头脑中运行这段代码，预测输出将是什么样子；然后在 R 中运行代码并检查你的预测。

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = island)  
) +  
  geom_point() +  
  geom_smooth(se = FALSE)
```

10. 这两张图看起来会不同吗？为什么相同/为什么不同？

```
ggplot(  
  data = penguins,  
  mapping = aes(x = flipper_length_mm, y = body_mass_g)  
) +  
  geom_point()  
  
ggplot() +  
  geom_point(  
    data = penguins,  
    mapping = aes(x = flipper_length_mm, y = body_mass_g)  
) +  
  geom_smooth(  
    data = penguins,  
    mapping = aes(x = flipper_length_mm, y = body_mass_g)  
)
```

## 1.3 ggplot2 调用

随着从入门部分学习的深入，我们将过渡到更简洁的 ggplot2 代码表达式。到目前为止，我们已经非常明确了，简化代码对你的学习是非常有帮助的：

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point()
```

通常，函数的前一两个参数非常重要，你应该牢记它们。在 `ggplot()` 中，前两个参数是 `data` 和 `mapping`，在本书的剩余部分，我们将不再提供这些参数的名称。这样既节省打字时间，又可通过减少额外文字，更容易地看出不同图之间的差异。这是一个非常重要的编程关注点，我们将在章节 ?? 中再次讨论。

将前面的图更简洁地重写为：

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```

在将来，您还将了解管道 `|>`，它将允许你使用以下命令创建该图形：

```
penguins |>
  ggplot(aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```

## 1 数据可视化

### 1.4 可视化分布

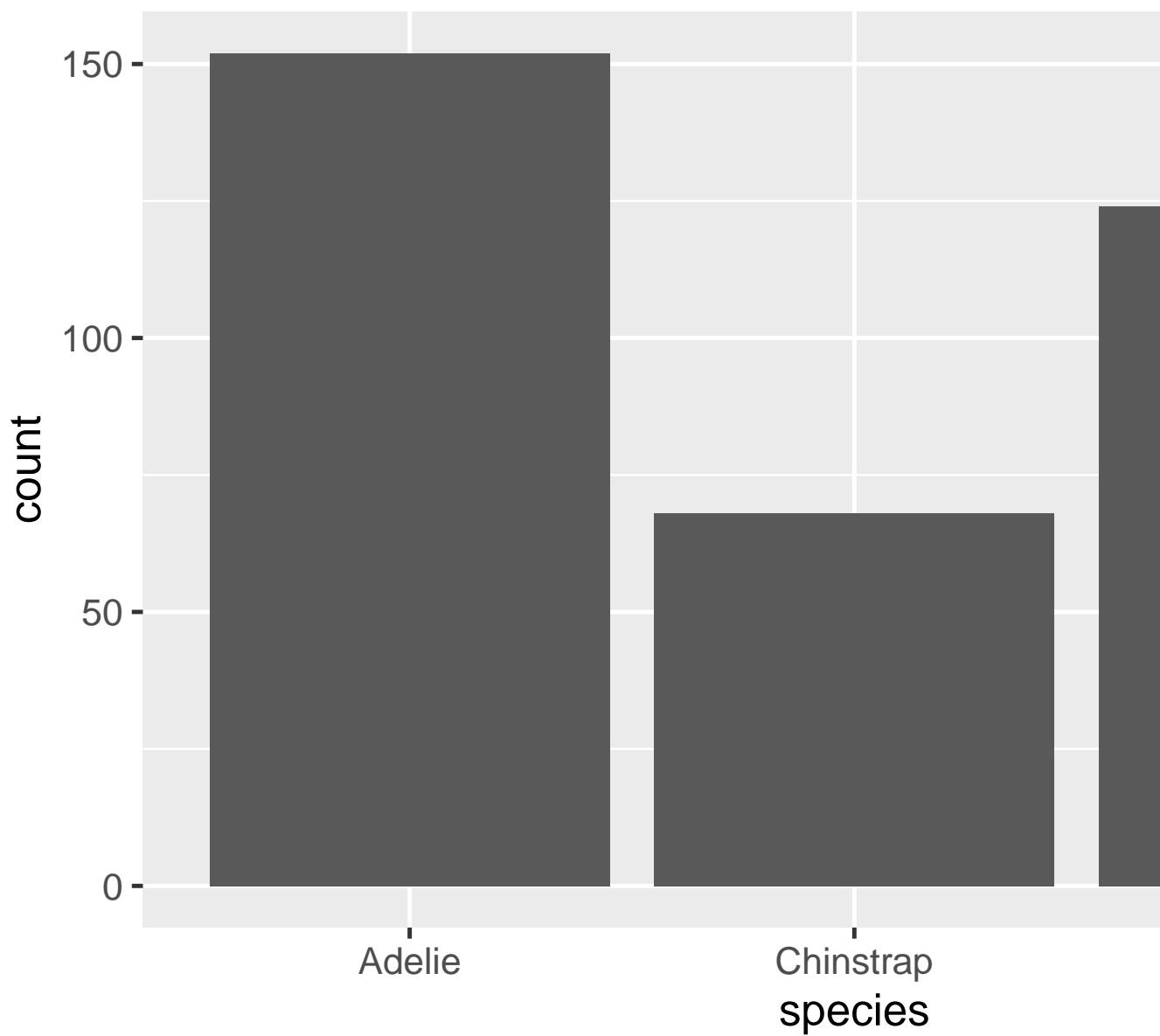
如何可视化变量的分布取决于变量的类型：分类还是数值。

#### 1.4.1 分类变量

如果一个变量只能取一小组值中的一个，那么它就是分类变量。要检查分类变量的分布，可以使用条形图。条形图的高度显示了每个 `x` 值的观测次数。

```
ggplot(penguins, aes(x = species)) +  
  geom_bar()
```

#### 1.4 可视化分布

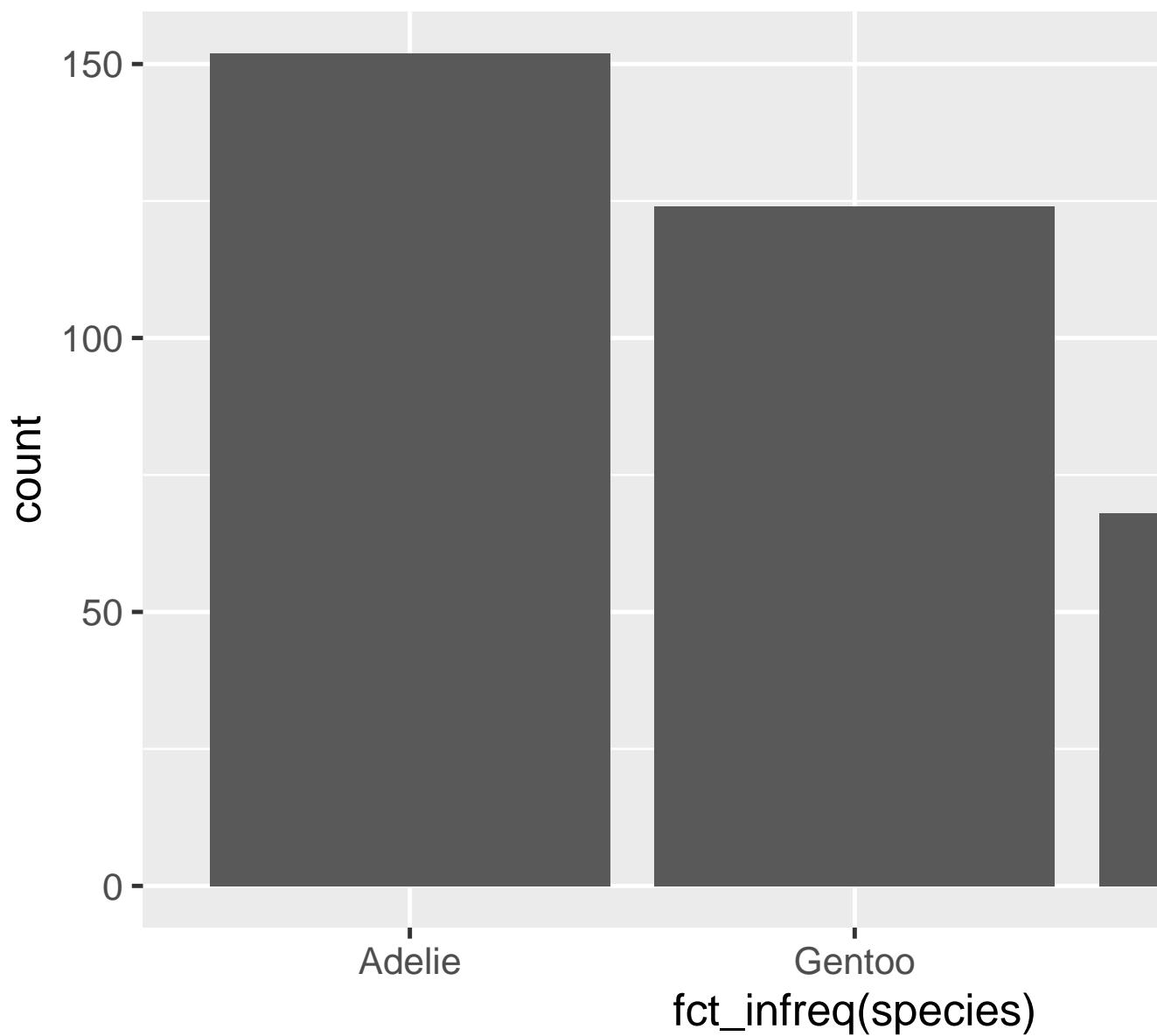


## 1 数据可视化

在具有非有序水平的分类变量的条形图中，如上面提到的企鹅物种，通常更可取思路的是根据它们的频率重新排序条形图。这样做需要将变量转换为因子 (R 如何处理分类数据)，然后重新排序该因子的水平。

```
ggplot(penguins, aes(x = fct_infreq(species))) +  
  geom_bar()
```

#### 1.4 可视化分布



## 1 数据可视化

您将在 @sec-factors 中学到更多关于因子和处理因子的函数（如上面用到的 `fct_infreq()`）。

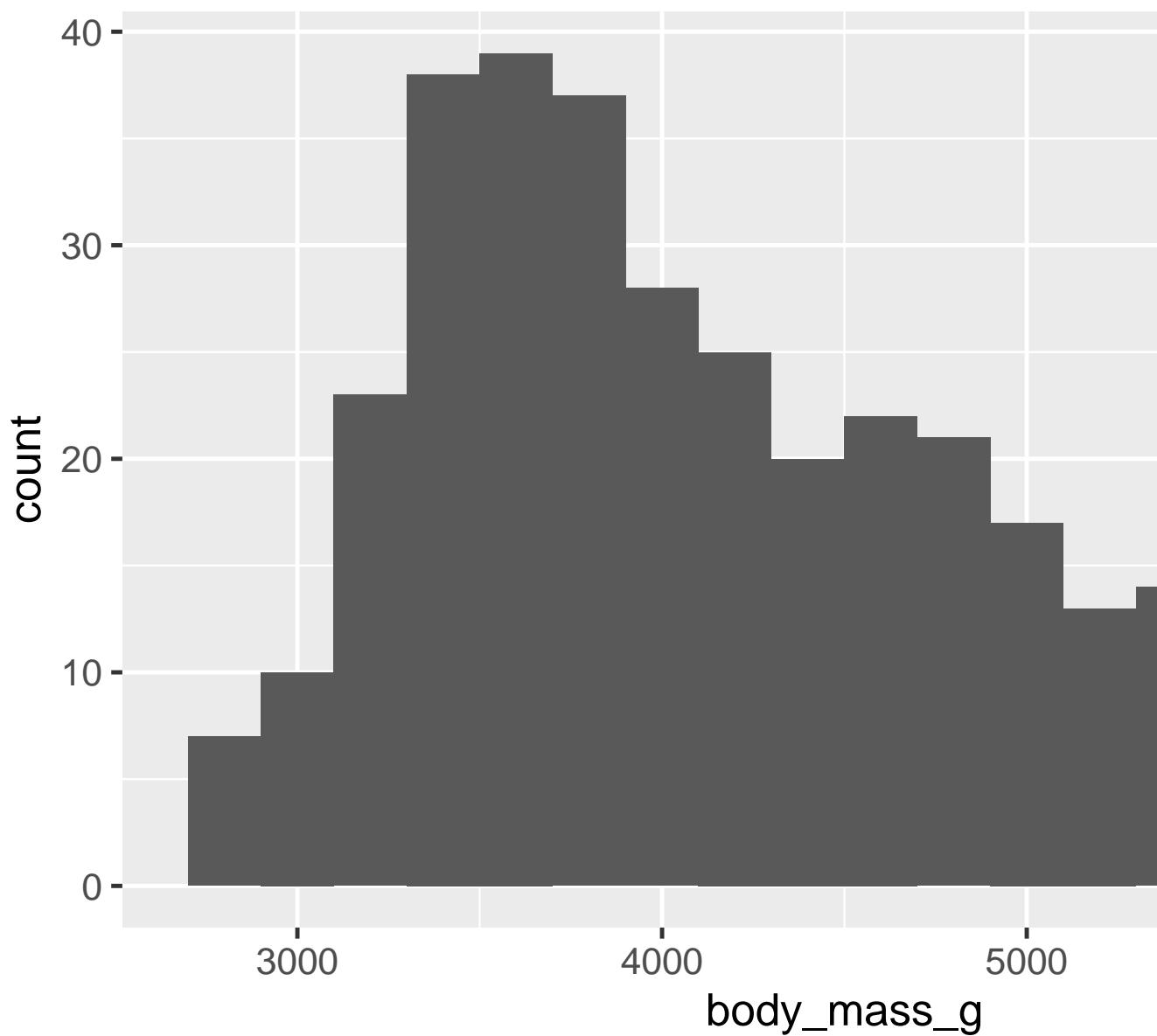
### 1.4.2 数值变量

如果一个变量可以在一个大的数值范围内取值，并且对这些数值进行加、减或取平均数是有意义的，那么这个变量就是数值型（或定量型）的。数值型变量可以是连续的，也可以是离散的。

一个常用于连续变量分布的可视化方法是直方图。

```
ggplot(penguins, aes(x = body_mass_g)) +  
  geom_histogram(binwidth = 200)
```

#### 1.4 可视化分布



## 1 数据可视化

直方图将 x 轴分成等间距的区间（或称为“箱”），然后使用条形的高度来显示落在每个区间内的观察值的数量。在上面的图中，最高的条形表示有 39 个观察值的 body\_mass\_g 值在 3,500 克到 3,700 克之间，这是条形的左右边缘。

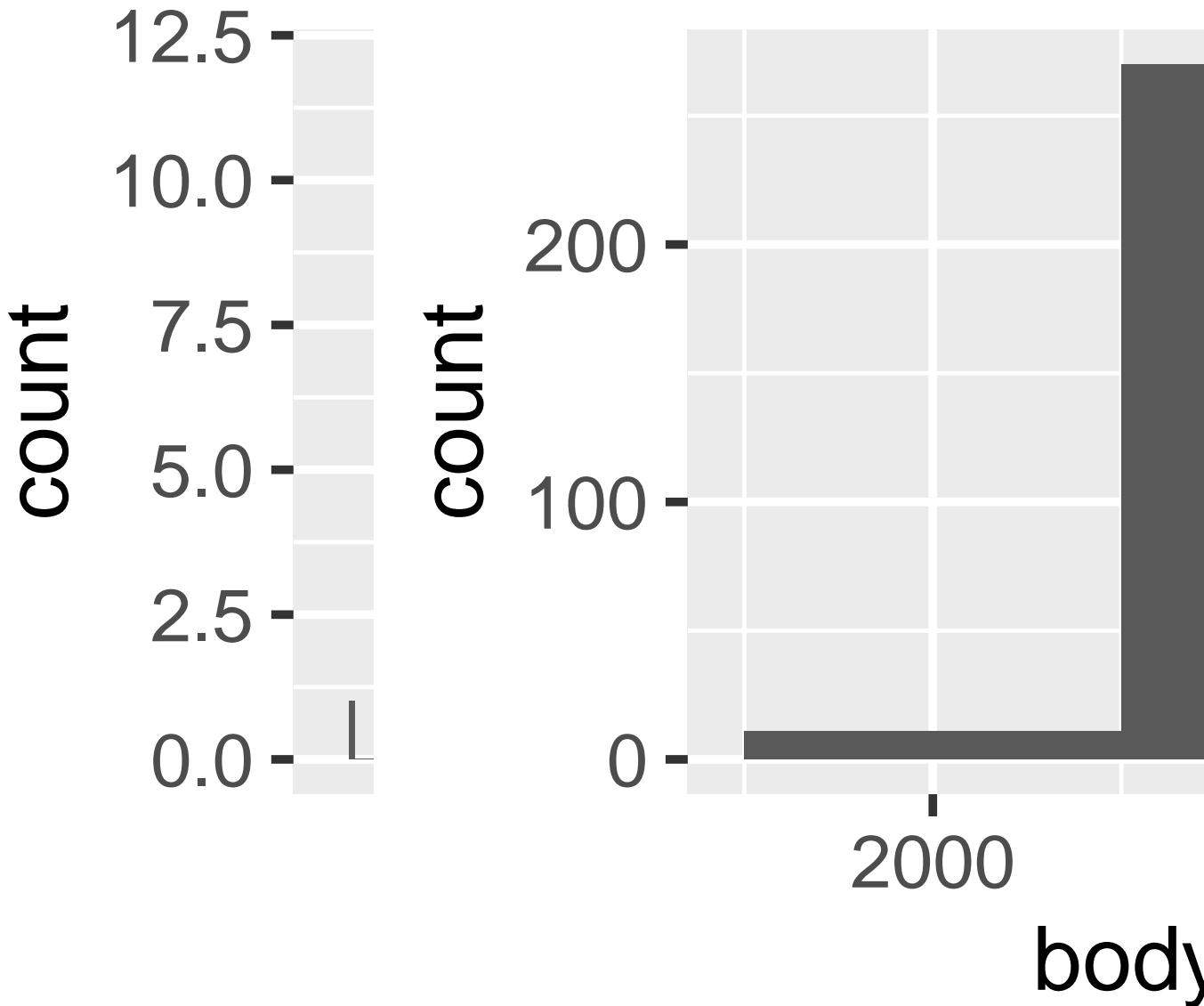
你可以使用 binwidth 参数来设置直方图中区间的宽度，这个参数是以 x 变量的单位来衡量的。当使用直方图时，你应该探索不同的区间宽度，因为不同的区间宽度可能会揭示不同的模式。在下面的图中，区间宽度为 20 太窄了，导致条形太多，使得难以确定分布的形状。类似地，区间宽度为 2,000 太高了，导致所有数据只被分到三个条形中，也难以确定分布的形状。区间宽度取 200 达到了合理的平衡。

```
ggplot(penguins, aes(x = body_mass_g)) +  
  geom_histogram(binwidth = 20)  
ggplot(penguins, aes(x = body_mass_g)) +  
  geom_histogram(binwidth = 2000)
```

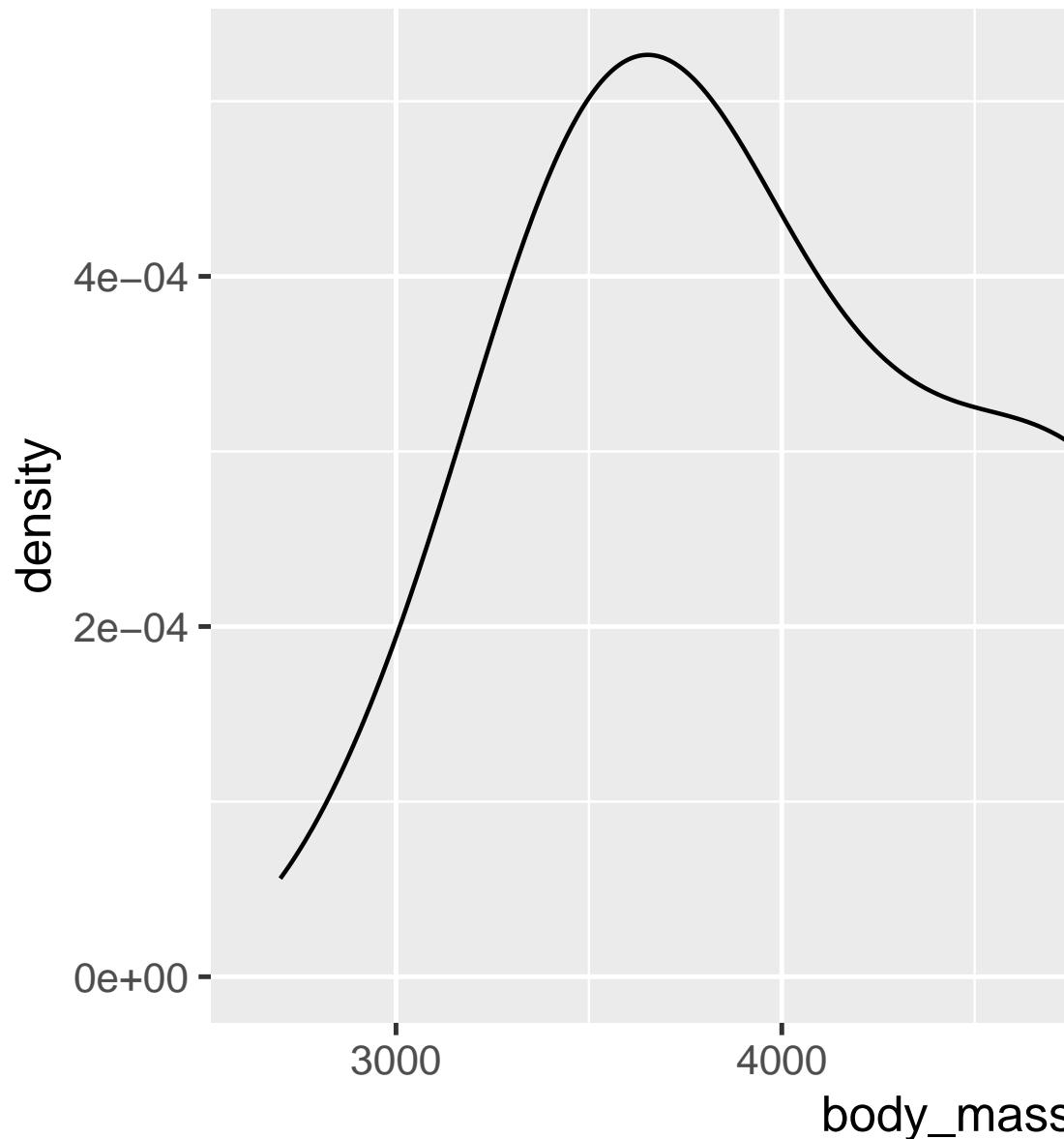
数值变量分布的另一种可视化方式是密度图。密度图是直方图的平滑版本，是连续数据的实用替代方案，特别是当数据来自一个潜在的平滑分布时。我们不会深入了解 `geom_density()` 如何估计密度（你可以在函数文档中了解更多），但我们可通过一个类比来解释如何绘制密度曲线。想象一个由木块制成的直方图，然后想象你在直方图上放下一根煮熟的面条，面条在木块上披挂的形状可以看作是密度曲线的形状。与直方图相比，它显示的细节较少，但可以更容易地快速了解分布的形状，特别是关于众数和偏态。

```
ggplot(penguins, aes(x = body_mass_g)) +  
  geom_density()  
#> Warning: Removed 2 rows containing non-finite outside the scale range  
#> (`stat_density()`).
```

#### 1.4 可视化分布



## 1 数据可视化



### 1.4.3 练习

1. 做一个企鹅种类的条形图，把物种分配给 y。这个图有什么不同？
2. 下面两个图有什么不同？哪种美学（颜色或填充）对改变条的颜色更有用？

```
ggplot(penguins, aes(x = species)) +
  geom_bar(color = "red")

ggplot(penguins, aes(x = species)) +
  geom_bar(fill = "red")
```

3. `geom_histogram()` 中的参数 `bins` 起什么作用？
4. 在加载 `tidyverse` 包后，对 `diamonds` 数据集中的 `carat` 变量制作一个直方图。尝试使用不同的区间宽度。哪种区间宽度揭示了最有趣的模式？

## 1.5 可视化关系

为了可视化关系，我们需要将至少两个变量映射到图的美学上。在下面的内容中，你将了解用于可视化两个或多个变量之间关系的常用绘图以及用于创建它们的几何图形。

To visualize a relationship we need to have at least two variables mapped to aesthetics of a plot. In the following sections you will learn about commonly used plots for visualizing relationships between two or more variables and the geoms used for creating them.

## 1 数据可视化

### 1.5.1 数值变量和分类变量

为了可视化数值变量和分类变量之间的关系，我们可以使用并列箱线图。箱线图是一种描述分布的位置度量指标（百分位数）的视觉简写形式，它还有助于识别潜在的异常值。如图 ?? 所示，每个箱线图包括：

- 一个箱子，表示数据中间一半的范围，即从分布的 25% 百分位数到 75% 百分位数的距离，这个距离称为四分位距 (interquartile range, IQR)。在箱子的中间有一条线，表示分布的中位数，即 50% 百分位数。这三条线可以让你感受到分布的离散程度以及分布是否关于中位数对称或偏向一侧。
- 视觉点，用于显示距离箱子边缘超过 1.5 倍 IQR 的观测值。这些异常点是不寻常的，因此单独绘制。
- 一条线（或称为“胡须”），从箱子的每个末端延伸出去，直到分布中最远的非异常值点。

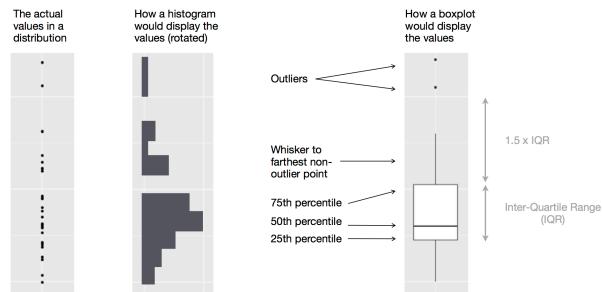


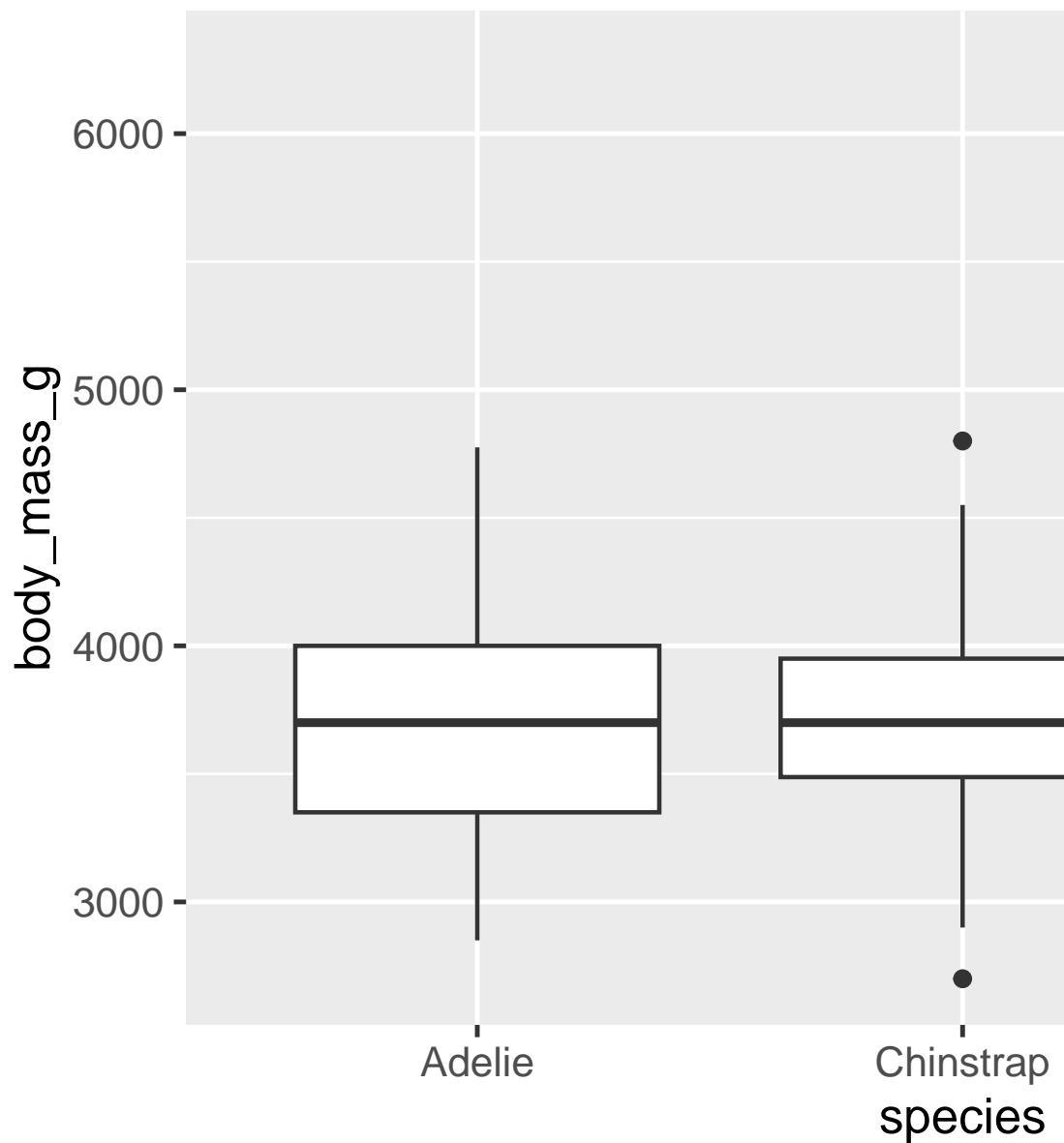
图 1.1: Diagram depicting how a boxplot is created.

让我们用 `geom_boxplot()` 来看看按物种分组的企鹅体重的分布：

## 1.5 可视化关系

```
ggplot(penguins, aes(x = species, y = body_mass_g)) +  
  geom_boxplot()
```

1 数据可视化

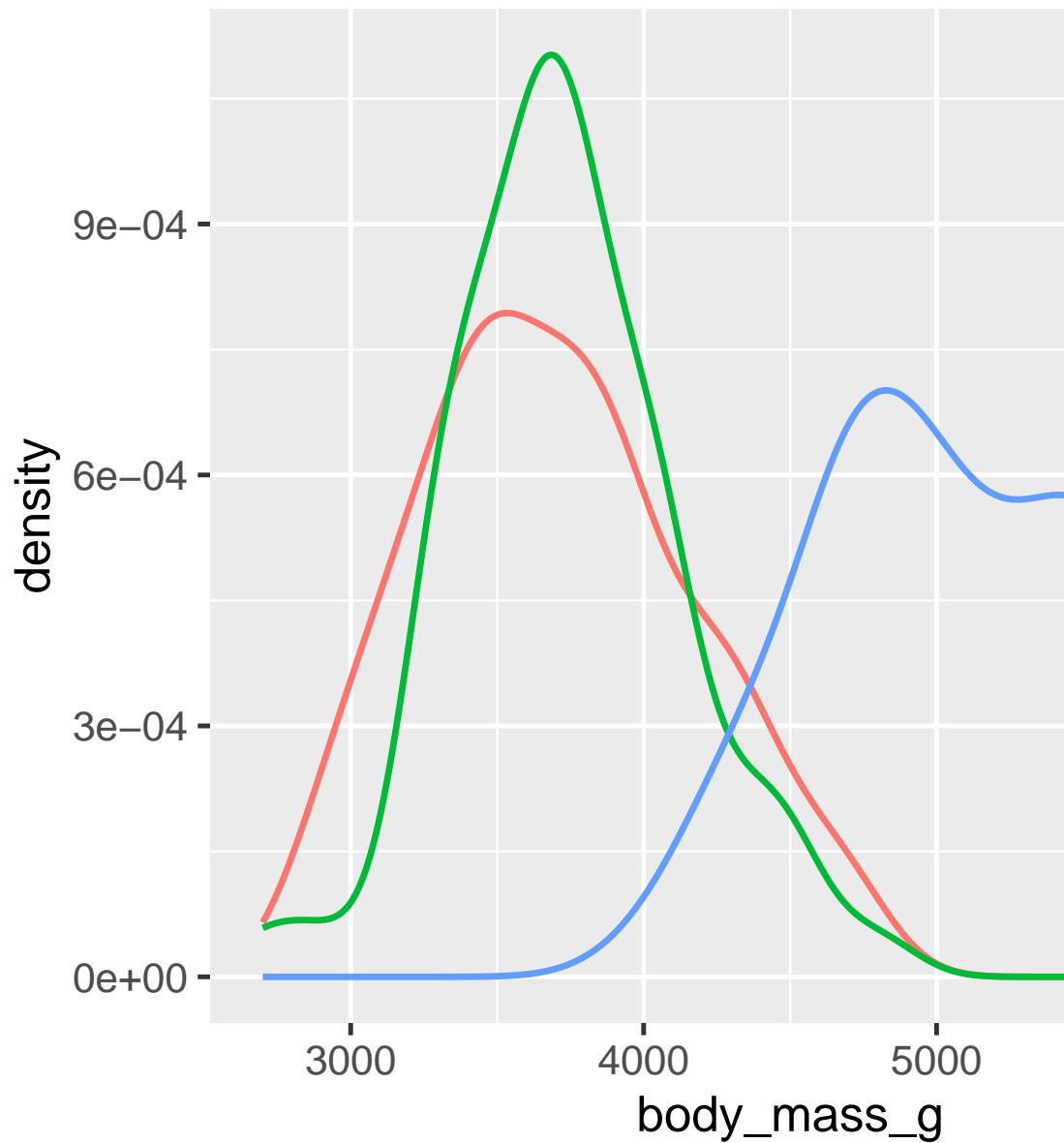


## 1.5 可视化关系

或者，使用 `geom_density()` 绘制密度图。

```
ggplot(penguins, aes(x = body_mass_g, color = species)) +  
  geom_density(linewidth = 0.75)
```

## 1 数据可视化



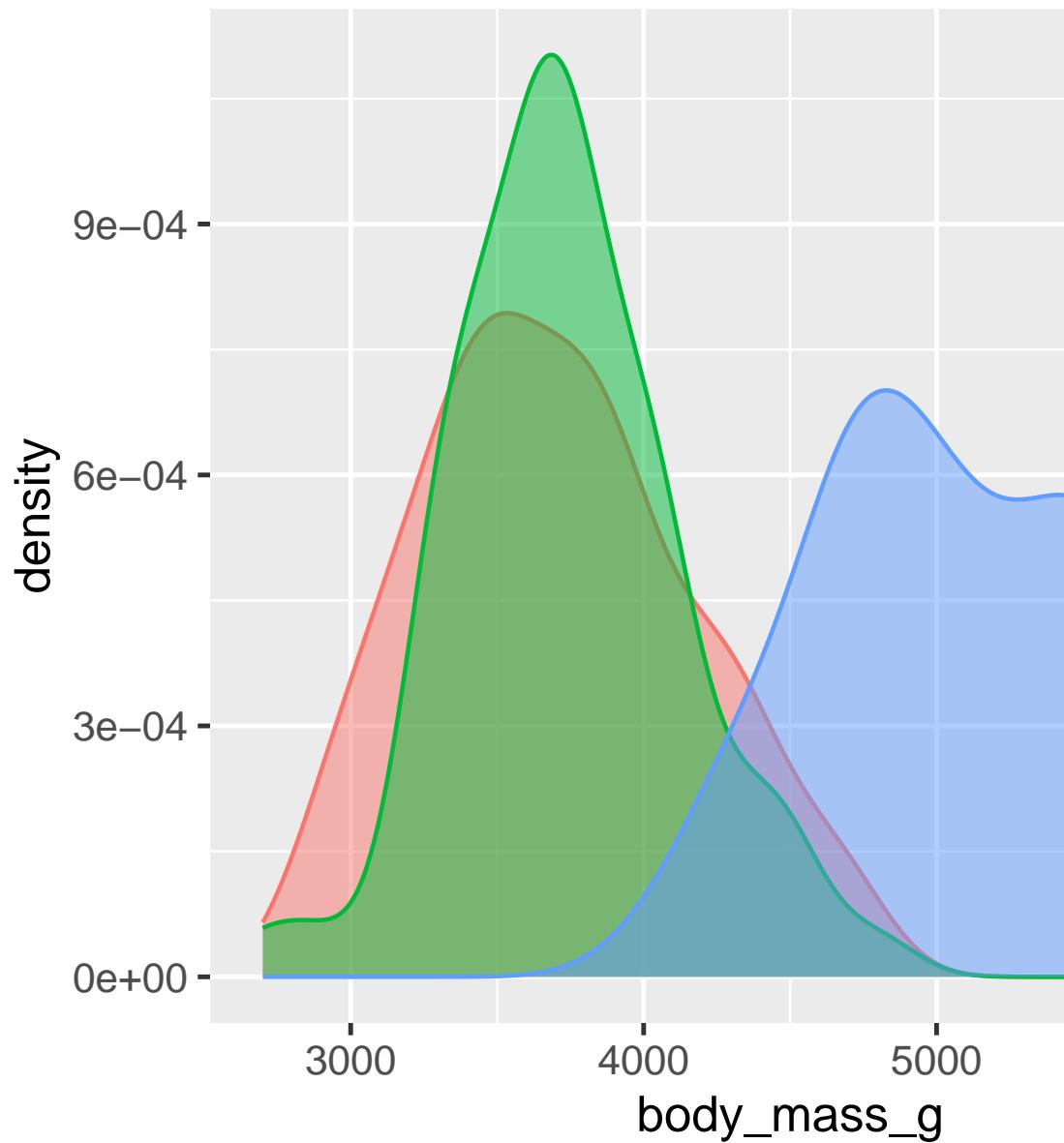
## 1.5 可视化关系

我们还使用了 `linewidth` 参数来定制线条的粗细，以便使它们在背景中更加突出。

此外，我们可以将物种映射到颜色和填充的美学特性，并使用 `alpha` 美学特性为填充的密度曲线添加透明度。这个美学特性取值范围在 0（完全透明）和 1（完全不透明）之间。在下图中，它被设置为 0.5。

```
ggplot(penguins, aes(x = body_mass_g, color = species, fill = species)) +  
  geom_density(alpha = 0.5)
```

## 1 数据可视化



注意我们在这里使用的术语：

- 如果想让美学所代表的视觉属性根据变量的值而变化，我们可以将变量映射到美学。
- 否则，设置该美学特性的值。

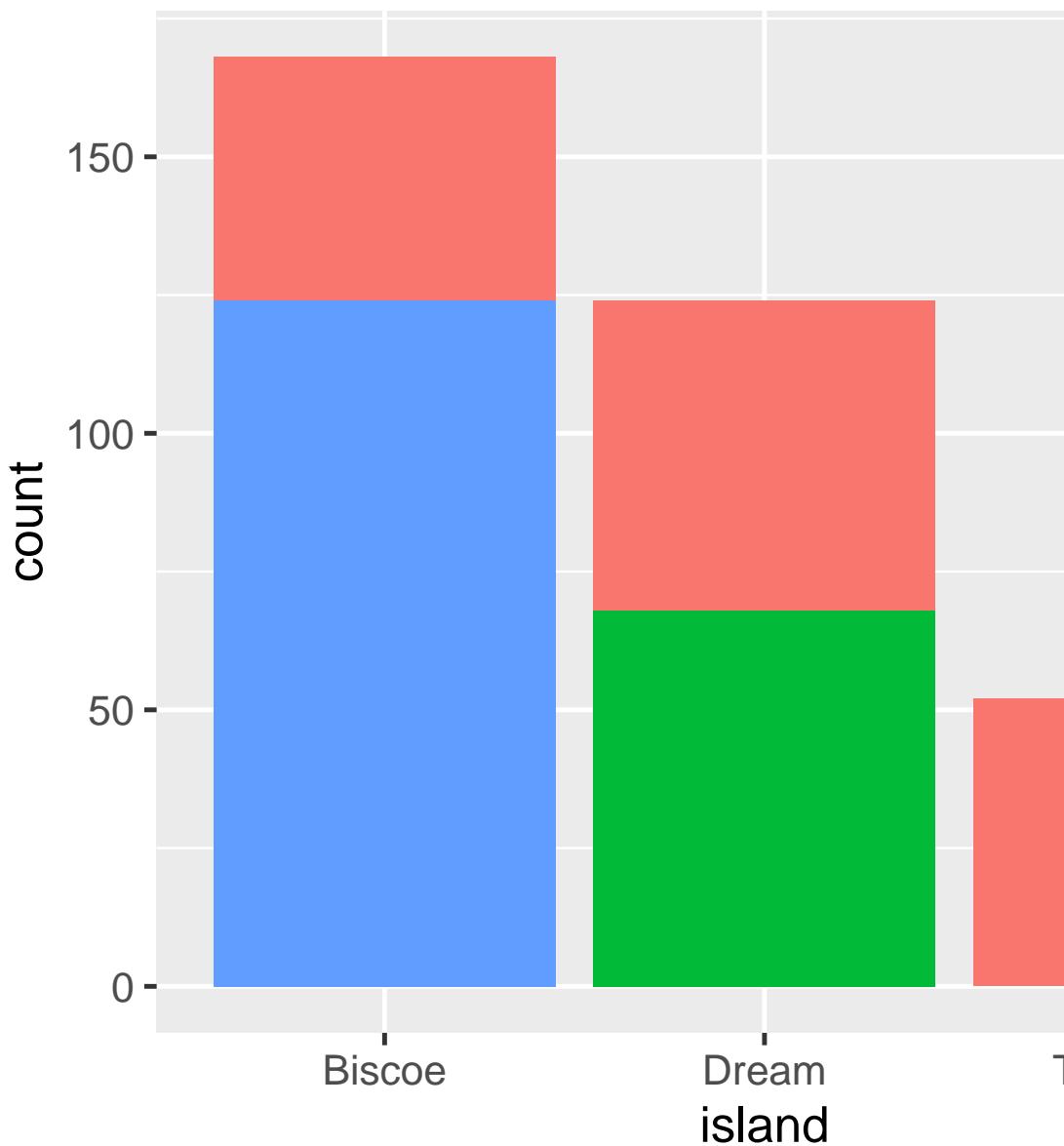
### 1.5.2 两个分类变量

我们可以使用堆叠条形图来可视化两个分类变量之间的关系。例如，下面的两个堆叠条形图都显示了岛屿和物种之间的关系，或者具体地说，显示了每个岛屿内物种的分布。

第一张图显示了每个岛屿上每种企鹅的频数。频数图显示，每个岛上的 Adelies 企鹅数量相等。但是，我们无法很好地感知每个岛屿内部的比例平衡。

```
ggplot(penguins, aes(x = island, fill = species)) +  
  geom_bar()
```

## 1 数据可视化

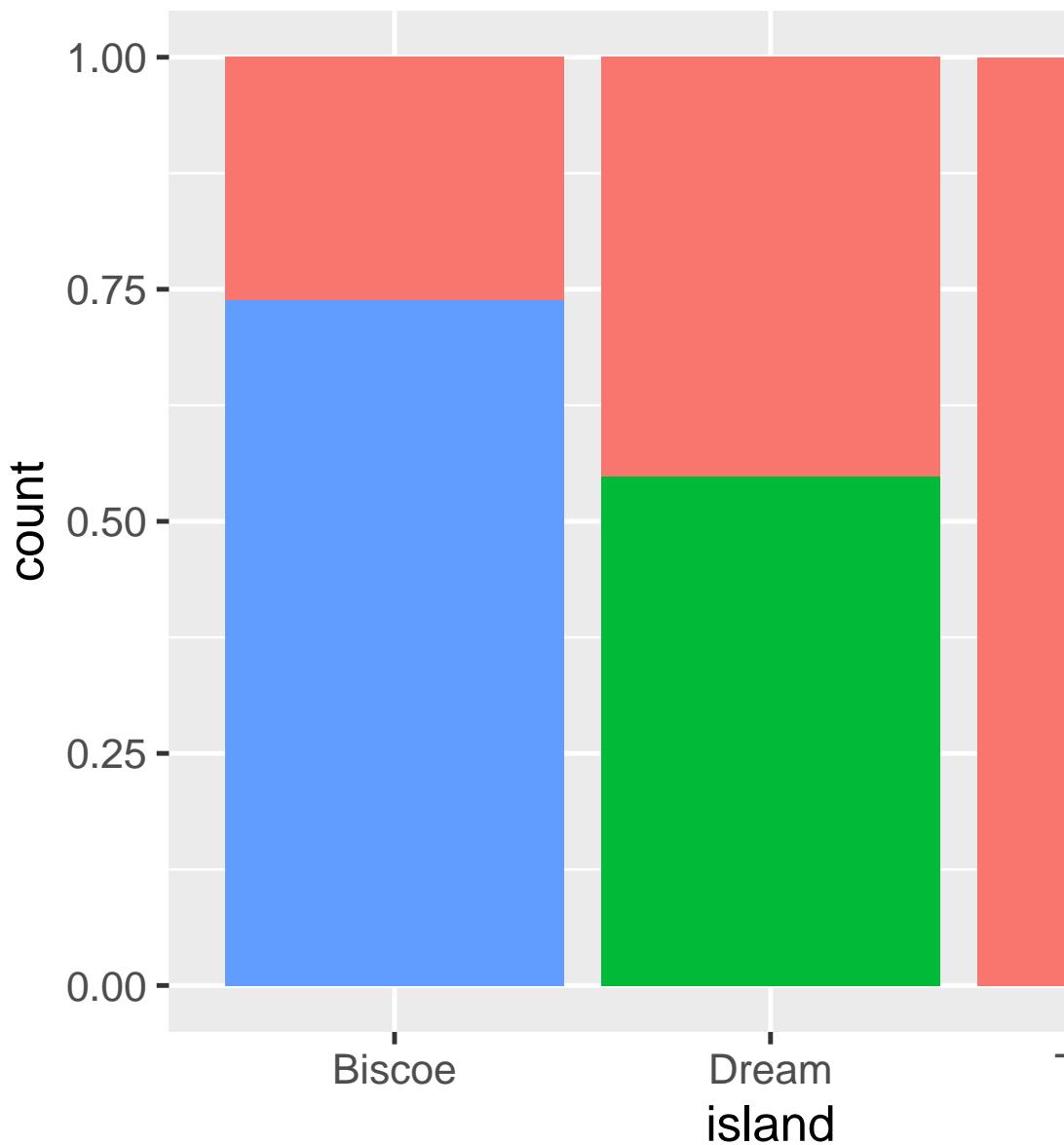


## 1.5 可视化关系

第二个图是通过在 geom 中设置 `position = "fill"` 创建的相对频数图，它对于比较不同岛屿上的物种分布更有用，因为它不受岛屿之间企鹅数量不等的影响。根据此图，我们可以看到 Gentoo 企鹅都生活在 Biscoe 岛上，约占该岛屿企鹅的 75%，Chinstrap 企鹅都生活在 Dream 岛上，约占该岛屿企鹅的 50%，而 Adelie 企鹅生活在所有三个岛屿上，并且 Torgersen 岛上都是 Torgersen 企鹅。

```
ggplot(penguins, aes(x = island, fill = species)) +  
  geom_bar(position = "fill")
```

## 1 数据可视化



## 1.5 可视化关系

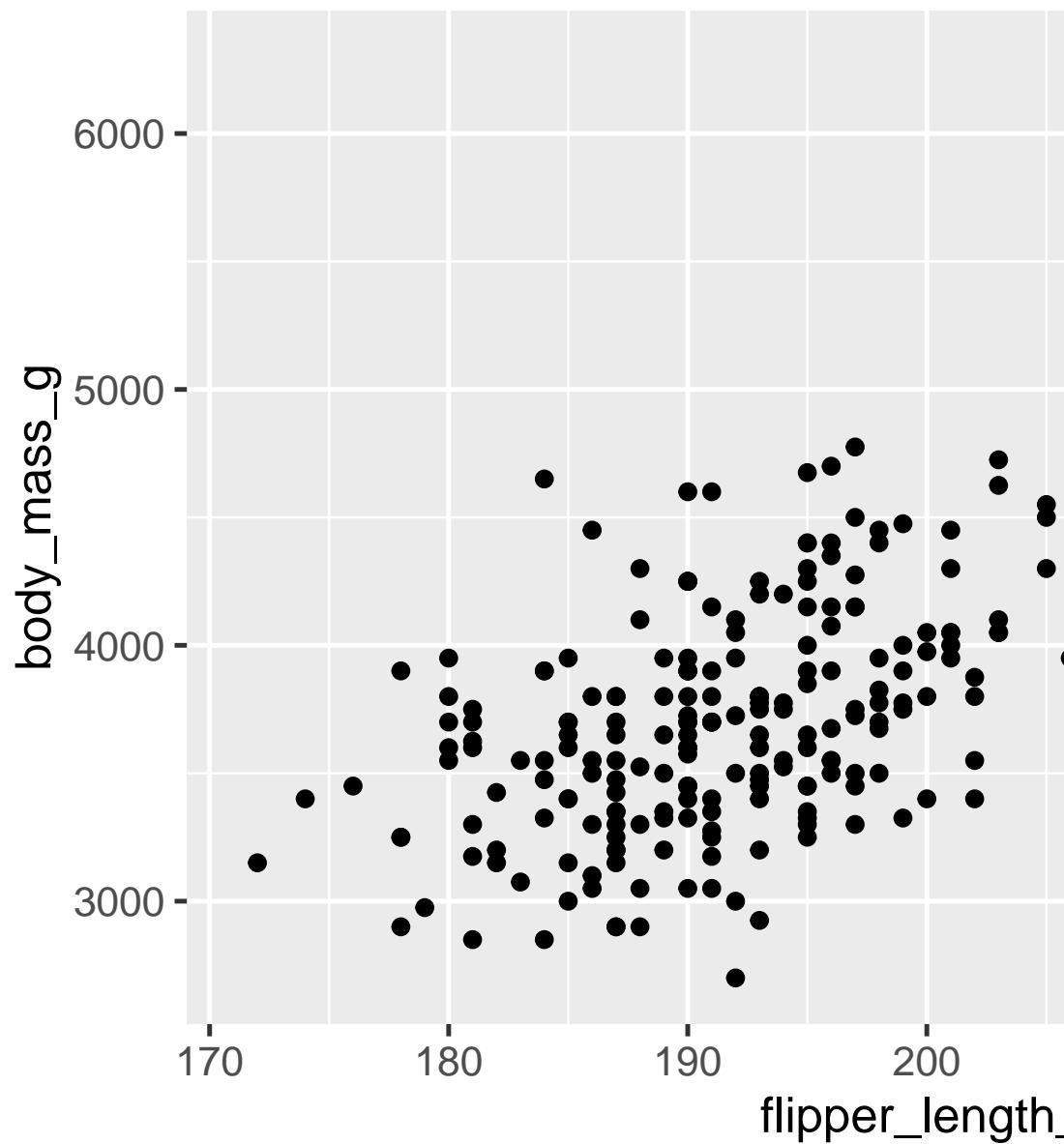
在创建这些条形图时，我们将要被分割成条形的变量映射到 `x` 美学上，而将改变条形内部颜色的变量映射到 `fill` 美学上。

### 1.5.3 两个数值变量

到目前为止，你已经学习了散点图（用 `geom_point()` 创建）和平滑曲线（用 `geom_smooth()` 创建），用于可视化两个数值变量之间的关系。散点图可能是用于可视化两个数值变量间关系的最常用图形。

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +  
  geom_point()
```

## 1 数据可视化

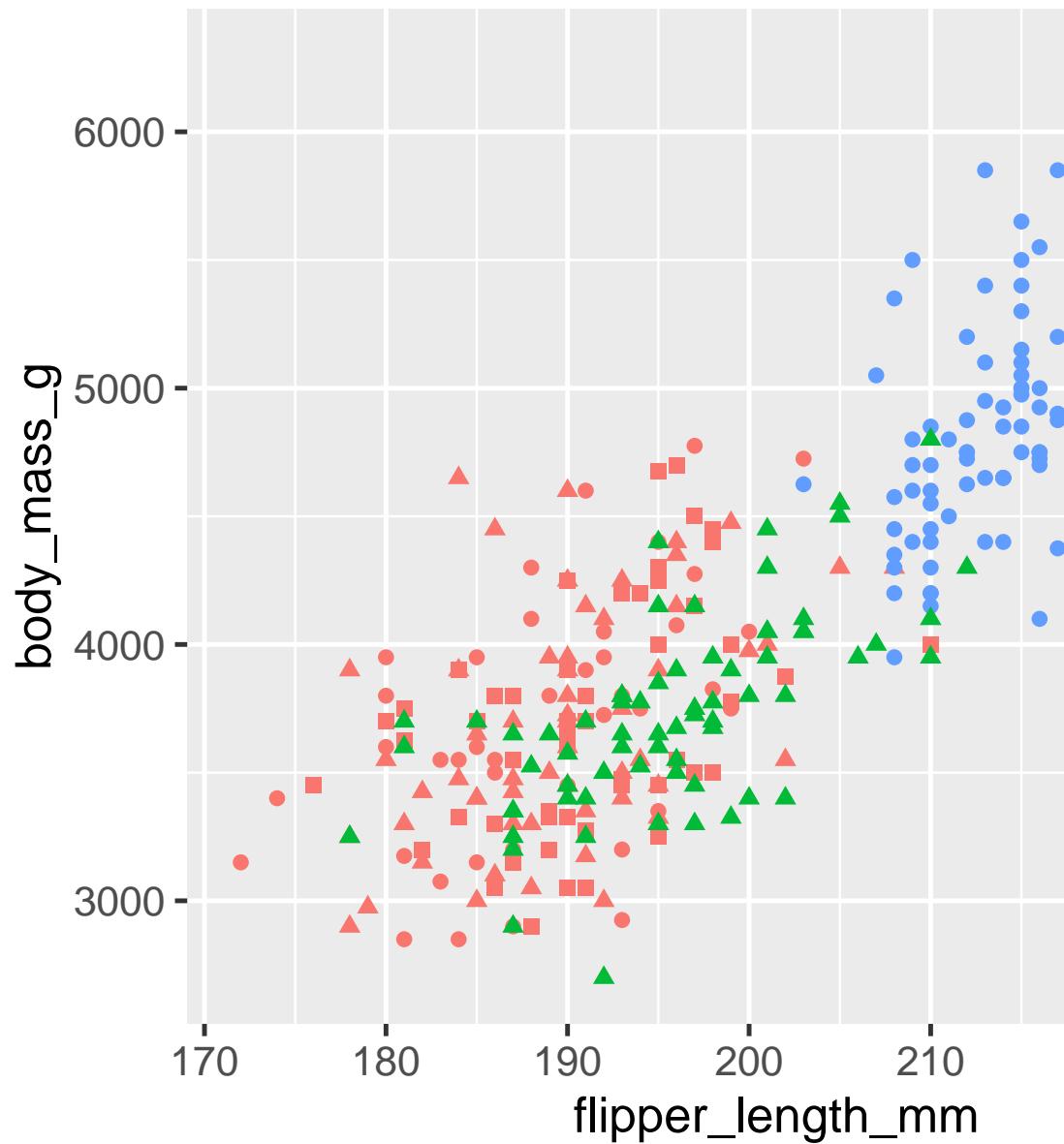


#### 1.5.4 三个或更多变量

正如我们在 @sec-adding-aesthetics-layers 中看到的，我们可以通过将更多变量映射到额外的美学特性来将它们融入图表中。例如，在下面的散点图中，点的颜色代表物种，而点的形状代表岛屿。

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +  
  geom_point(aes(color = species, shape = island))
```

## 1 数据可视化



## 1.5 可视化关系

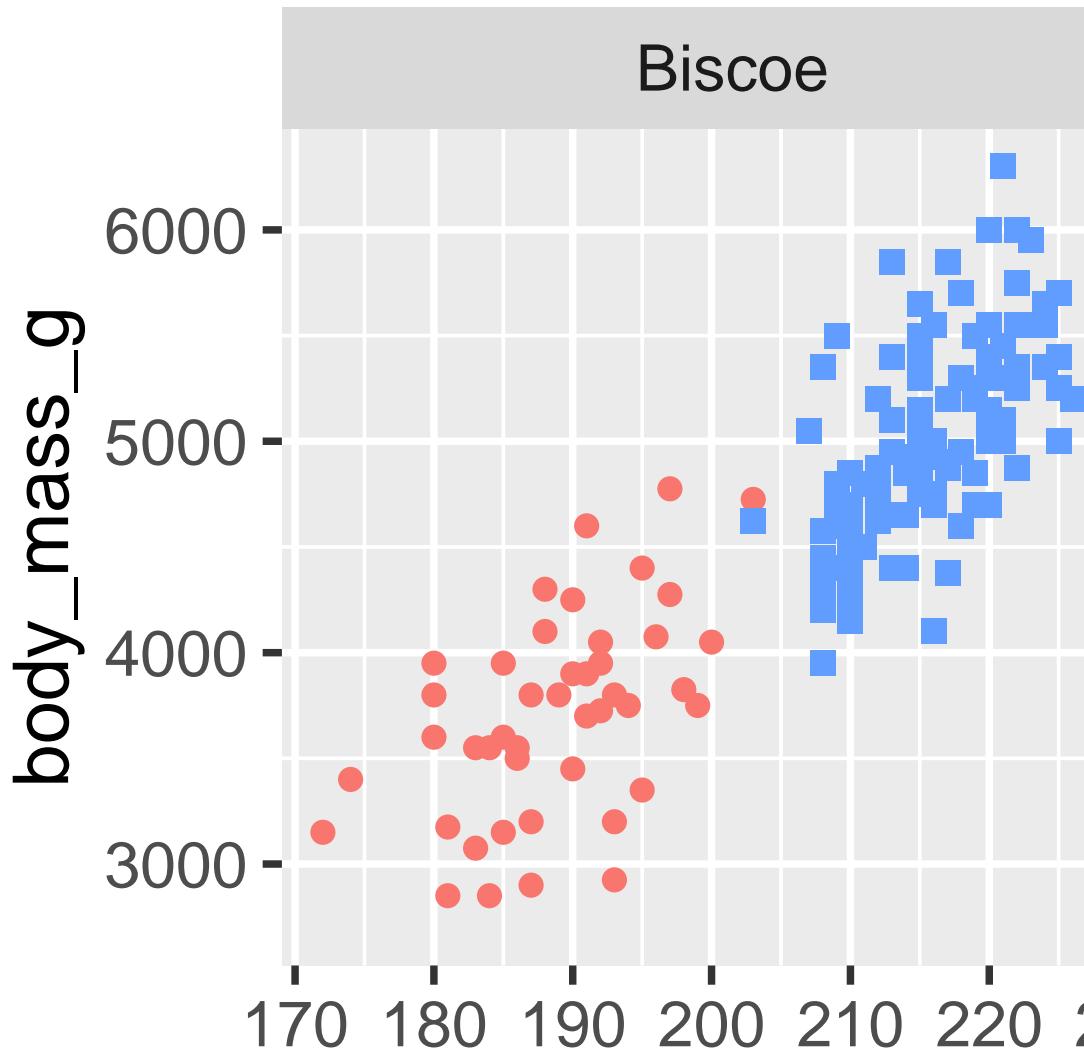
然而，在图表中添加过多的美学映射会使它变得杂乱无章且难以理解。另一种特别适用于分类变量的方法是将图形拆分为多个分面（**facets**），即每个分面显示数据的一个子集。

要通过单个变量将图表拆分为分面，请使用 `facet_wrap()`。`facet_wrap()` 的第一个参数是一个公式（formula）<sup>3</sup>，通过 ~ 后跟变量名来创建这个公式，传递给 `facet_wrap()` 的变量应该是分类变量。

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +  
  geom_point(aes(color = species, shape = species)) +  
  facet_wrap(~island)
```

---

<sup>3</sup>这里的“formula”是由 ~ 创建的事物的名称，而不是“equation”的同义词。



在 @sec-layers 中, 你将学习许多其他用于可视化变量分布以及它们之间关系的几何对象 (geoms)。

### 1.5.5 练习

1. `ggplot2` 包中捆绑的 `mpg` 数据框包含了美国环境保护局收集的 234 个观测值, 涵盖了 38 种汽车型号。`mpg` 中的哪些变量是分类变量? 哪些变量是数值变量? (提示: 键入`?mpg` 以读取数据集的文档。) 当你运行 `mpg` 时, 如何查看这些信息?
2. 使用 `mpg` 数据框制作 `hwy` 与 `displ` 的散点图。接下来, 将第三个数值变量映射到 `color`, 然后映射到 `size`, 再同时映射到 `color` 和 `size`, 最后映射到 `shape`。这些美学特性在分类变量与数值变量上表现有何不同?
3. 在 `hwy` 与 `displ` 的散点图中, 如果将第三个变量映射到 `linewidth`, 会发生什么?
4. 如果你将同一个变量映射到多个美学特性上会发生什么?
5. 制作 `bill_depth_mm` 与 `bill_length_mm` 的散点图, 并按 `species` 对点进行着色。通过按物种着色, 可以揭示这两个变量之间的关系是什么? 如果按 `species` 分面又会如何?
6. 为什么以下代码会生成两个独立的图例? 你如何修复它以合并这两个图例?

```
ggplot(
  data = penguins,
  mapping = aes(
```

## 1 数据可视化

```
x = bill_length_mm, y = bill_depth_mm,  
color = species, shape = species  
)  
) +  
geom_point() +  
labs(color = "Species")
```

7. 创建以下两个堆叠条形图。第一个图可以回答哪个问题? 第二个图可以回答哪个问题?

```
ggplot(penguins, aes(x = island, fill = species)) +  
  geom_bar(position = "fill")  
ggplot(penguins, aes(x = species, fill = island)) +  
  geom_bar(position = "fill")
```

## 1.6 保存图形

一旦你创建了图形, 你可能想将其从 R 中导出并保存为图像, 以便在其他地方使用。这就是 `ggsave()` 函数的作用, 它会将最近创建的图保存到磁盘上:

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +  
  geom_point()  
ggsave(filename = "penguin-plot.png")
```

这会将图保存到工作目录, 您会在 `@sec-workflow-scripts-projects` 中更多地了解这个概念。

## 1.7 常见问题

如果你没有指定 `width` 和 `height`, 则将从当前绘图设备的尺寸中获取它们。为了代码的可重复性, 你应该指定它们。可以在文档中了解更多关于 `ggsave()` 的信息。

然而, 一般来说, 我们推荐你使用 Quarto 来组合你的最终报告, Quarto 是一个可重复的创作系统, 它允许你将代码和文本穿插在一起, 并自动将你的图包含在报告中。你将在 `@sec-quarto` 中了解更多关于 Quarto 的信息。

### 1.6.1 练习

1. 运行以下代码行。这两个图中哪一个被保存为 `mpg-plot.png`? 为什么?

```
ggplot(mpg, aes(x = class)) +  
  geom_bar()  
ggplot(mpg, aes(x = cty, y = hwy)) +  
  geom_point()  
ggsave("mpg-plot.png")
```

2. 你需要在上面的代码中更改什么以将图形保存为 PDF 而不是 PNG? 如何知道在 `ggsave()` 中可以保存哪些类型的图像文件?

## 1.7 常见问题

当你开始运行 R 代码时, 你很可能会遇到问题。不用担心, 每个人都会遇到这样的问题。我们已经写了多年的 R 代码, 但每天我们还是会写出第一次尝试时不工作的代码!

## 1 数据可视化

首先，仔细比较你正在运行的代码和书中的代码。R 非常挑剔，一个放错位置的字符可能就会产生截然不同的结果。确保每个（都有一个匹配的），每个双引号" 都有另一个" 配对。有时你运行代码后什么都不会发生。检查控制台左侧：如果是一个 +，这意味着 R 认为你没有输入完整的表达式，它在等待你完成它。在这种情况下，通常很容易通过按 ESC 键来中止当前命令的处理，然后重新开始。

在创建 ggplot2 图形时，一个常见的问题是将加号 + 放在错误的位置：它必须放在行的末尾，而不是开头。换句话说，确保你没有写出这样的代码：

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

如果你仍然无法解决问题，尝试使用帮助功能。你可以在控制台中运行`?function_name` 来获取任何 R 函数的帮助，或者在 RStudio 中突出显示函数名并按 F1 键。如果帮助信息看起来不太有用，不要担心，直接跳到示例部分，寻找与你试图做的事情相匹配的代码。

如果这没有帮助，请仔细阅读错误消息。有时答案就隐藏在其中！但是当你刚开始学习 R 时，即使答案在错误消息中，你可能还不知道如何理解它。另一个很好的工具是 Google：尝试搜索错误消息，因为很可能有人遇到过同样的问题，并在网上得到了帮助。

## 1.8 小结

在本章中，你学习了使用 ggplot2 进行数据可视化的基础知识。我们首先介绍了 ggplot2 的基本原理：可视化是一种将你的数据中的变量映射到诸如位置、颜色、大小和形状等美学属性的过程。然后，你学习了如何逐层增加复杂性并改进

## 1.8 小结

你的图表的呈现方式。你还学习了如何利用额外的美学映射和/或通过将图表分割成多个小图 (faceting) 来可视化单个变量的分布以及两个或多个变量之间的关系。

在本书中, 我们将反复使用可视化, 并在需要时介绍新的技术; 同时在 @sec-layers 到 @sec-communication 中更深入地探讨使用 ggplot2 创建可视化。

掌握了可视化的基础后, 我们将在下一章中稍微转换一下方向, 给你一些实用的工作流程建议。我们在本书的这一部分穿插了工作流程建议和数据科学工具, 因为这将帮助你在编写越来越多的 R 代码时保持组织有序。



## 2 工作流程：基础

现在你已经有了运行 R 代码的经验。虽然我们没有给你很多细节，但你肯定已经了解了一些基本知识，否则你会沮丧地把这本书扔掉！当你开始用 R 编程时，心情沮丧是很自然的，因为 R 对标点符号是如此的挑剔，一个字符不合适都可能导致它报错。当你感到沮丧时，你要安慰自己，这种经历是普遍的、暂时的，每个人都会遇到这种情况，克服它的唯一方法就是继续努力。

在我们进一步讨论之前，让我们确保你在运行 R 代码方面有一个坚实的基础，并且你知道一些最有用的 RStudio 特性。

### 2.1 编程基础

让我们回顾一下之前我们忽略的一些基础知识，以便让您尽快学会绘图。你可以用 R 来做基本的数学计算：

```
1 / 200 * 30
#> [1] 0.15
(59 + 73 + 2) / 3
#> [1] 44.66667
sin(pi / 2)
#> [1] 1
```

## 2 工作流程：基础

可以使用赋值操作符创建新对象 <-:

```
x <- 3 * 4
```

注意，x 的值没有被打印出来，它只是被保存起来了。如果要查看该值，请在控制台中键入 x。

可以用 c() 将多个元素组合成一个向量:

```
primes <- c(2, 3, 5, 7, 11, 13)
```

向量的基本运算应用于向量的每一个元素:

```
primes * 2  
#> [1] 4 6 10 14 22 26  
primes - 1  
#> [1] 1 2 4 6 10 12
```

所有创建对象的 R 语句，亦即赋值语句，都有相同的形式:

```
object_name <- value
```

当你读这段代码时，在心里默念“对象名获得了值”。

你会进行大量的赋值操作，而 <- 打起来比较费劲，你可以使用 RStudio 的快捷键来节省时间: Alt + - (减号)。请注意，RStudio 会自动在 <- 周围添加空格，这是一种良好的写代码习惯。代码在美好的日子里读起来也可能很令人痛苦（试试 giveyoureyesabreak），所以给你的眼睛休息一下，使用空格吧。

## 2.2 注释

R 会忽略该行中 `#` 之后的任何文本。这允许你编写注释，即 R 会忽略但人类可以阅读的文本。我们有时会在示例中包含注释，以解释代码正在做什么。

注释可以帮助您简要描述下面代码的功能。

```
# create vector of primes
primes <- c(2, 3, 5, 7, 11, 13)

# multiply primes by 2
primes * 2
#> [1] 4 6 10 14 22 26
```

像这样简短的代码段，可能不需要为每一行代码都留下注释。但是，随着编写的代码变得越来越复杂，注释可以帮助你（及你的合作者）节省很多时间来弄清楚代码是做什么的。

使用注释来解释你编写代码的原因（why），而不是方法（how）或内容（what）。通过仔细阅读代码，总是有可能弄清楚代码的方法和内容的，尽管这可能会很繁琐。如果你在注释中描述了每一步，然后修改了代码，你需要记得同时更新注释，否则当你将来再次查看这段代码时，可能会感到困惑。

弄清楚为什么这么做往往更加困难，甚至是不可能的。例如，`geom_smooth()` 函数有一个名为 `span` 的参数，它控制曲线的平滑度，较大的值会产生更平滑的曲线。假设你决定将 `span` 的值从默认的 0.75 改为 0.9：未来的读者很容易理解发生了什么，但除非你在注释中注明你的思考过程，否则没有人会理解你为什么要改变默认值。

## 2 工作流程：基础

对于数据分析代码，使用注释来解释你的整体策略和步骤，并在遇到重要见解时记录下来。这些知识是无法仅从代码本身重新获取的。

### 2.3 对象名称

对象名必须以字母开头，并且只能包含字母、数字、`_` 和 `.`。你希望对象名是描述性的，因此需要为多个单词采用一种约定。我们推荐使用蛇形命名法(`snake_case`)，即使用`_`分隔小写单词。

```
i_use_snake_case  
otherPeopleUseCamelCase  
some.people.use.periods  
And_aFew.People_RENOUNCEconvention
```

当讨论章节 `??` 中的代码风格时，我们将再次回到对象名。

你可以通过输入一个对象的名字来检查它：

```
x  
#> [1] 12
```

下面是另外一个赋值：

```
this_is_a_really_long_name <- 2.5
```

要检查这个对象，请尝试 RStudio 的补全功能：输入“`this`”，按 TAB 键，添加字符，直到有一个唯一的前缀，然后按回车键。

## 2.4 函数调用

假设你犯了一个错误，`this_is_a_really_long_name` 的值应该是 3.5 而不是 2.5，你可以使用另一个键盘快捷键来帮助你修正它。例如，按↑键来调出你刚才输入的最后一个命令并进行编辑。或者输入“this”，然后按 Cmd/Ctrl + ↑ 来列出输入过的所有以这些字母开头的命令，使用箭头键来导航。然后按 Enter 键重新输入命令，将 2.5 改为 3.5 并重新运行。

再来一个赋值：

```
r_rocks <- 2^3
```

试着检查一下：

```
r_rock  
#> Error: object 'r_rock' not found  
R_rocks  
#> Error: object 'R_rocks' not found
```

这说明你和 R 之间的存在隐含约定：R 会为你完成繁琐的计算，但相应地，你的指令必须完全精确。如果不这样做，你可能会收到一个错误提示，说未找到你正在查找的对象。拼写很重要，如你在输入 `r_rock` 时可能指的是 `r_rocks`，但 R 无法读懂你的想法。大小写也很重要，如你输入 `R_rocks` 时可能指的是 `r_rocks`，但 R 同样无法读懂你的想法。

## 2.4 函数调用

R 有大量的内置函数，可以这样调用：

## 2 工作流程：基础

```
function_name(argument1 = value1, argument2 = value2, ...)
```

让我们尝试使用 `seq()` 函数来生成一系列有规律的数字，同时也学习 RStudio 的一些更实用的特性。输入 `se` 并按下 TAB 键，这时会弹出一个窗口显示可能的补全选项。通过输入更多内容（如 `q`）来明确指定 `seq()`，或者使用 ↑/↓ 箭头来选择。注意弹出的浮动提示，它会提醒你函数的参数和用途。如果你想要更多帮助，按下 F1 键，在右下角窗格的帮助选项卡中获取所有详细信息。

当选择了想要的函数后，再次按下 TAB 键，RStudio 会为你添加匹配的左括号（和右括号）。输入第一个参数的名字 `from`，将其设置为等于 1，然后，输入第二个参数的名字 `to`，并将其设置为等于 10。最后，按下回车键。

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

在函数调用中我们经常省略前几个参数的名字，所以这一行代码也可以这样写：

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

输入以下代码，注意到 RStudio 在成对的引号上也提供了类似的帮助：

```
x <- "hello world"
```

引号和括号总是成对出现。R Studio 会尽力帮助你，但仍然有可能出错并导致不匹配。如果发生这种情况，R 会显示续行字符 +：

## 2.5 练习

```
> x <- "hello
+
+
```

+ 告诉你 R 正在等待更多的输入，它认为你还没有完成。通常，这意味着你忘记了输入一个" 或者一个)。你可以添加缺失的配对符号，或者按下 ESCAPE 键来中止表达式并重新尝试。

请注意，右上角窗格中的环境（Environment）选项卡显示了你创建的所有对象：

The screenshot shows the RStudio interface with the 'Environment' tab selected. The 'Values' section displays the following variables:

Variables	Values
primes	num [1:6] 2 3 5 7 11 13
r_rocks	8
this_is_a_really_long_name	2.5
x	12

## 2.5 练习

1. 为什么这段代码不能正常运行？

```
my_variable <- 10
my_variable
#> Error in eval(expr, envir, enclos): object 'my_variable' not found
```

仔细看！这可能看起来像是无意义的练习，但能训练你的大脑注意到即使是微小的差异，这在编程时也会让你有所受益。

2. 微调以下每个 R 命令，以便它们可以正确运行：

## 2 工作流程：基础

```
library(tidyverse)

ggplot(dTA = mpg) +
  geom_point(mapping = aes(x = displ y = hwy)) +
  geom_smooth(method = "lm")
```

3. 按下 Option + Shift + K / Alt + Shift + K，会发生什么？你如何通过菜单到达同样的地方？
4. 重新看一一下来自小节 ?? 的一个练习。运行以下代码行。哪个图形被保存为 `mpg-plot.png`？为什么？

```
my_bar_plot <- ggplot(mpg, aes(x = class)) +
  geom_bar()
my_scatter_plot <- ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point()
ggsave(filename = "mpg-plot.png", plot = my_bar_plot)
```

## 2.6 小结

现在你已经学习了更多关于 R 代码如何工作的知识以及一些提示。在将来重新使用代码时这些提示能帮助你更好地理解它。在下一章，我们将继续数据科学之旅，向你介绍 dplyr，它是 tidyverse 的一个包，可以帮助你转换数据，例如选择重要的变量、筛选感兴趣的行还是计算汇总统计量。

# 3 数据转换

## 3.1 引言

可视化是生成新见解的重要工具，但很少能刚好得到你所需的确切形式的数据来制作你想要的图。通常，你需要创建一些新的变量或汇总统计量来用数据回答你的问题；或者你可能只是想重命名变量或重新排序观测值，以便数据更容易处理。在本章中，你将学习如何做到这些（及更多）。本章将介绍使用 **dplyr** 包和 2013 年从纽约市出发的航班数据集进行数据转换。

本章的目标是为你概述转换数据框的所有关键工具。我们将从对数据框的行和列进行操作的函数开始，然后再次回到讨论管道（pipe），这是一个重要的工具，用于组合动词。接下来，我们将介绍处理分组的能力。我们将以一个研究案例结束本章，该案例研究展示了这些函数的实际应用；当我们开始深入研究特定类型的数据（例如数字、字符串、日期）时，我们将在后续的章节中更详细地讨论这些函数。

### 3.1.1 必要条件

本章将重点讨论 **dplyr** 包，它是 **tidyverse** 的另一个核心成员。我们将使用来自 **nycflights13** 包的数据来说明 **dplyr** 包的关键理念，并使用 **ggplot2** 来帮助我们

### 3 数据转换

理解数据。

```
library(nycflights13)
library(tidyverse)
#> -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
#> v dplyr     1.1.4      v readr     2.1.5
#> v forcats   1.0.0      v stringr   1.5.1
#> v ggplot2   3.5.0      v tibble    3.2.1
#> v lubridate  1.9.3      v tidyrr    1.3.1
#> v purrr    1.0.2
#> -- Conflicts -----
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()   masks stats::lag()
#> i Use the conflicted package (<a href="http://conflicted.r-lib.org/">http://conflicted.r-lib.org/) to force all co
```

请注意加载 tidyverse 时显示的冲突消息。它告诉你 dplyr 覆盖了 R 基础包中的一些函数。如果你在加载 dplyr 后想使用这些函数的基础版本，你需要使用它们的全名: stats::filter() 和 stats::lag()。到目前为止，我们主要忽略了函数来自哪个包，因为大多数情况下这并不重要。但是，知道包名可以帮助你找到帮助和相关的函数，所以当我们需要精确地知道一个函数来自哪个包时，我们将使用与 R 相同的语法: packagename::functionname()。

#### 3.1.2 nycflights13

为了探索基本的 dplyr 操作，我们将使用 nycflights13::flights。此数据集包含所有 r format(nrow(nycflights13::flights), big.mark = ",")2013 年从纽约出发的航班。这些数据来自美国[Bureau of Transportation Statistics](#)，并记录在?flights。

### 3.1 引言

```
flights  
#> # 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     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  
#> # i 336,770 more rows  
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

flights 是一个 tibble (tibble 是 tidyverse 使用的一种特殊类型的数据框)，用于避免一些常见的意外情况。tibble 和数据框之间最重要的区别在于它们的输出方式；tibble 是为了大型数据集设计的，因此它们只显示前几行和能够在屏幕上显示的列。有几种方法可以查看所有内容。如果你正在使用 RStudio，最方便的可能是 View(flights)，这会打开一个可以滚动和筛选的交互式视图。另外你可以使用 print(flights, width = Inf) 来显示所有列，或者使用 glimpse() 函数：

```
glimpse(flights)  
#> Rows: 336,776  
#> Columns: 19  
#> $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013~  
#> $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~  
#> $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~  
#> $ dep_time        <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 55~
```

### 3 数据转换

```
#> $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 60~  
#> $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, ~  
#> $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 8~  
#> $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 8~  
#> $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, ~  
#> $ carrier         <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6"~  
#> $ flight          <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301~  
#> $ tailnum         <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N~  
#> $ origin          <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LG~  
#> $ dest            <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IA~  
#> $ air_time        <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149~  
#> $ distance        <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 73~  
#> $ hour             <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6~  
#> $ minute           <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59~  
#> $ time_hour        <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-0~
```

在这两种视图中，变量名后面跟着缩写，这些缩写表示每个变量的类型：  
<int> 是整数的缩写，<dbl> 是双精度浮点数（也称为实数）的缩写，<chr> 是字符（也称为字符串）的缩写，<dttm> 是日期时间的缩写。这些缩写很重要，因为你在一列上执行的操作在很大程度上取决于它的“类型”。

#### 3.1.3 dplyr 基础

下面将学习 dplyr 的主要操作（即函数），这些函数能帮你解决绝大多数的数据处理问题。但在讨论它们各自的差异之前，值得一提的是它们的共同点：

1. 第一个参数始终是一个数据框；

## 3.2 行

2. 后续参数通常使用变量名（不带引号）来描述要操作的列；
3. 输出始终是一个新的数据框。

因为每个函数都擅长做一件事，所以解决复杂问题通常需要组合多个函数，我们将使用管道操作符 `|>` 来实现这一点。我们将在 @sec-the-pipe 更详细地讨论管道操作符。简言之，管道操作符将其左侧的内容传递给其右侧的函数，因此 `x |> f(y)` 等同于 `f(x, y)`，而 `x |> f(y) |> g(z)` 等同于 `g(f(x, y), z)`，`|>` 最简单的发音是“then”。这使得即使你还没有学习细节，也有可能理解下面的代码：

```
flights |>  
  filter(dest == "IAH") |>  
  group_by(year, month, day) |>  
  summarize(  
    arr_delay = mean(arr_delay, na.rm = TRUE)  
  )
```

dplyr 的函数根据作用对象不同分为四组：行 (rows)、列 (columns)、组 (groups) 或表 (tables)。下面将学习有关行、列和组的最重要函数，然后回到 @sec-joins 中探讨作用于表的合并操作。让我们开始吧！

## 3.2 行

对数据集的行进行操作的最重要的函数是 `filter()`，它影响行的去留而不改变它们的顺序；而函数 `arrange()` 改变行的顺序而不影响行的去留。这两个函数都只影响行，列保持不变。我们还将讨论 `distinct()`，它找出具有唯一值的行，但与 `arrange()` 和 `filter()` 不同，它还可以选择性地修改列。

### 3 数据转换

#### 3.2.1 filter()

Filter() 允许你根据列的值保留行<sup>1</sup>。第一个参数是数据框，第二个及随后的参数是保留该行必须为真的条件。例如，我们可以找出所有迟到超过 120 分钟 (2 小时) 的航班：

```
flights |>
  filter(dep_delay > 120)
#> # A tibble: 9,723 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       848        1835      853     1001          1950
#> 2 2013     1     1       957        733       144     1056          853
#> 3 2013     1     1      1114        900       134     1447          1223
#> 4 2013     1     1      1540       1338       122     2020          1829
#> 5 2013     1     1      1815       1325       290     2120          1542
#> 6 2013     1     1      1842       1422       260     1958          1538
#> # i 9,717 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

除了使用 > (大于) 之外，还可以使用 >= (大于或等于)、< (小于)、<= (小于或等于)、== (等于) 和!= (不等于)。你还可以使用 & 或 | 来组合条件，表示“和”（检查两个条件是否都满足），或者使用 | 来表示“或”（检查任一条件是否满足）：

---

<sup>1</sup>稍后，您将学习 slice\_\*() 系列函数，它允许你根据行的位置选择行。

### 3.2 行

```
# Flights that departed on January 1
flights |>
  filter(month == 1 & day == 1)
#> # A tibble: 842 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
#> # i 836 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...

# Flights that departed in January or February
flights |>
  filter(month == 1 | month == 2)
#> # A tibble: 51,955 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
```

### 3 数据转换

```
#> # i 51,949 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

当您结合使用 | 和 == 时有一个很有用的快捷方式: %in%。它会保留变量等于右侧值之一的行:

```
# A shorter way to select flights that departed in January or February
flights |>
  filter(month %in% c(1, 2))
#> # A tibble: 51,955 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       1024
#> 5 2013     1     1      554         600      -6     812        837
#> 6 2013     1     1      554         558      -4     740        728
#> # i 51,949 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

我们将在 @sec-logicals 中详细讨论这些比较和逻辑运算符。

当你运行 filter() 时, dplyr 会执行筛选操作, 创建一个新的数据框, 然后输出它。它不会修改现有的 flights 数据集, 因为 dplyr 函数永远不会修改它们的输入。要保存结果, 你需要使用赋值运算符 <-:

```
jan1 <- flights |>
  filter(month == 1 & day == 1)
```

### 3.2.2 常见错误

从你刚开始学习 R 时，最容易犯的错误是在检验是否相等时使用 `=` 而不是 `==`。`filter()` 会告诉你发生了什么：

```
flights |>
  filter(month = 1)
#> Error in `filter()`:
#> ! We detected a named input.
#> i This usually means that you've used `=` instead of `==`.
#> i Did you mean `month == 1`?
```

另一个错误是像在英语中那样写“或”语句：

```
flights |>
  filter(month == 1 | 2)
```

这段代码“有效”，它不会抛出错误，但它没有按照你的期望去做，因为 `|` 首先检查条件 `month == 1`，然后检查条件 `2`，而检查 `2` 并不是一个合理的条件。我们将在小节 `??` 更多地了解这里发生了什么，以及为什么。

### 3 数据转换

#### 3.2.3 arrange()

arrange() 根据列的值改变行的顺序。它根据一个数据框和一组列名（或更复杂的表达式）来排序。如果你提供了多个列名，那么每个额外的列都将用于解决前面列中值的并列问题。例如，下面的代码按出发时间排序，出发时间分布在四个列中。我们首先得到最早的年份，然后在同一年份中，我们得到最早的月份，依此类推。

```
flights |>
  arrange(year, month, day, 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     1      517          515       2     830          813
#> 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         1023
#> 5  2013     1     1      554          600      -6     812          837
#> 6  2013     1     1      554          558      -4     740          728
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

你可以在 arrange() 内部对某一列使用 desc() 来根据该列的值以降序（从大到小）重新排序数据框。例如，这段代码将航班按延误时间从长到短排序：

```
flights |>
  arrange(desc(dep_delay))
#> # A tibble: 336,776 x 19
```

### 3.2 行

```
#>   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
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

注意，行数没有改变，我们只是对数据进行了排列，没有对其进行过筛选。

#### 3.2.4 distinct()

distinct() 在数据集中找出所有唯一的行，所以从技术上讲，它主要对行进行操作。然而，在大多数情况下，你会想要某些变量的不同组合，因此你也可以选择性地提供列名：

```
# Remove duplicate rows, if any
flights |>
  distinct()
#> # 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     1      517            515      2     830       819
#> 2 2013     1     1      533            529      4     850       830
```

### 3 数据转换

```
#> 3 2013 1 1 542 540 2 923 853
#> 4 2013 1 1 544 545 -1 1004 1023
#> 5 2013 1 1 554 600 -6 812 832
#> 6 2013 1 1 554 558 -4 740 723
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
#>
#> # Find all unique origin and destination pairs
flights |>
  distinct(origin, dest)
#> # A tibble: 224 x 2
#>   origin dest
#>   <chr>  <chr>
#> 1 EWR    IAH
#> 2 LGA    IAH
#> 3 JFK    MIA
#> 4 JFK    BQN
#> 5 LGA    ATL
#> 6 EWR    ORD
#> # i 218 more rows
```

或者,如果希望在筛选唯一行时保留其他列,您可以使用`.keep_all = TRUE`选项。

```
flights |>
  distinct(origin, dest, .keep_all = TRUE)
#> # A tibble: 224 x 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
```

### 3.2 行

```
#>   <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
#> # i 218 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

所有这些不同的航班都在 1 月 1 日并不是巧合：`distinct()` 会在数据集中找到第一个出现的唯一行，并丢弃其余的行。

如果你想要找到出现的次数，那么最好将 `distinct()` 替换为 `count()`，并使用 `sort = TRUE` 参数，你可以按照出现次数的降序排列它们。你将在小节 ?? 学到更多关于 `count` 的内容。

```
flights |>
  count(origin, dest, sort = TRUE)
#> # A tibble: 224 x 3
#>   origin dest     n
#>   <chr>  <chr> <int>
#> 1 JFK    LAX    11262
#> 2 LGA    ATL    10263
#> 3 LGA    ORD    8857
#> 4 JFK    SFO    8204
#> 5 LGA    CLT    6168
```

### 3 数据转换

```
#> 6 EWR      ORD      6100  
#> # i 218 more rows
```

#### 3.2.5 练习

1. 针对每个条件，在单个管道中查找所有符合以下条件的航班：
  - 到达延误两小时或更长时间
  - 飞往休斯敦 (IAH 或 HOU)
  - 由联合航空 (United)、美国航空 (American)，或达美航空 (Delta) 运营
  - 在夏季 (七月、八月和九月) 起飞
  - 到达延误超过两小时，但起飞不延误
  - 延误至少一小时，但在飞行中弥补了超过 30 分钟的时间
2. 对 flights 进行排序以找到出发延误时间最长的航班；找到早上最早起飞的航班；
3. 对 flights 进行排序以找到最快的航班。（提示：尝试在函数中包含数学计算。）
4. 2013 年的每一天都有航班吗？
5. 哪些航班飞行了最远的距离？哪些飞行了最短的距离？
6. 如果你同时使用 filter() 和 arrange()，你使用的顺序重要吗？为什么？思考一下结果以及这些函数需要执行多少工作。

## 3.3 列

有四个重要的函数，它们影响列而不改变行：`mutate()` 用于根据现有列创建新列，`select()` 用于更改列存留，`rename()` 用于更改列的名称，而 `relocate()` 用于更改列的位置。

### 3.3.1 `mutate()`

`mutate()` 的作用是添加根据现有列计算得到的新列。在数据转换章节中，你将学习一系列函数，这些函数可用于处理不同类型的变量。目前，我们将继续使用基础代数，这允许我们计算 `gain`（即延误航班在空中弥补了多少时间）以及每小时的速度：

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60
  )
#> # A tibble: 336,776 x 21
#>   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
```

### 3 数据转换

```
#> # i 336,770 more rows
#> # i 13 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

默认情况下，`mutate()` 会在数据集的右侧添加新列，但这使得我们很难看到数据集里发生了什么。可以使用`.before` 参数来将变量添加到左侧：

By default, `mutate()` adds new columns on the right hand side of your dataset, which makes it difficult to see what's happening here. We can use the `.before` argument to instead add the variables to the left hand side<sup>2</sup>:

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60,
    .before = 1
  )
#> # A tibble: 336,776 x 21
#>   gain speed year month day dep_time sched_dep_time dep_delay arr_time
#>   <dbl> <dbl> <int> <int> <int>     <int>       <int>     <dbl>     <int>
#> 1   -9  370.  2013     1     1      517        515         2     830
#> 2   -16  374.  2013     1     1      533        529         4     850
#> 3   -31  408.  2013     1     1      542        540         2     923
#> 4    17  517.  2013     1     1      544        545        -1    1004
#> 5    19  394.  2013     1     1      554        600        -6     812
#> 6   -16  288.  2013     1     1      554        558        -4     740
#> # i 336,770 more rows
#> # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>, ...
```

---

<sup>2</sup>Remember that in RStudio, the easiest way to see a dataset with many columns is `View()`.

### 3.3 列

. 是一个标志，表示.before 是函数的参数，而不是我们正在创建的第三个新变量的名称。你也可以使用.after 在某个变量之后添加新变量，在.before 和.after 中，你都可以使用变量名而不是位置。例如，我们可以在 day 之后添加新变量：

```
flights |>  
  mutate(  
    gain = dep_delay - arr_delay,  
    speed = distance / air_time * 60,  
    .after = day  
)
```

另外，你可以使用.keep 参数来控制保留哪些变量。一个特别有用的参数是”used”，它指定我们只保留在 mutate() 步骤中涉及或创建的列。例如，以下输出将仅包含 dep\_delay、arr\_delay、air\_time、gain、hours 和 gain\_per\_hour 这些变量。

```
flights |>  
  mutate(  
    gain = dep_delay - arr_delay,  
    hours = air_time / 60,  
    gain_per_hour = gain / hours,  
    .keep = "used"  
)
```

请注意，由于我们没有将上述计算的结果重新分配给 flights 数据框，新的变量 gain、hours 和 gain\_per\_hour 只会被输出出来，但不会被存储在一个数据框中。如果我们希望这些变量在未来的数据框中可用，我们应该仔细考虑是否要将结果重新分配给 flights，从而用更多的变量覆盖原始数据框，还是分配

### 3 数据转换

给一个新的对象。通常，正确的答案是创建一个新的对象，并为其命名以清晰地指示其内容，例如 `delay_gain`，但你也可能有充分的理由覆盖 `flights`。

#### 3.3.2 `select()`

获得包含数百甚至数千个变量的数据集并不罕见。在这种情况下，第一个挑战通常是仅关注你感兴趣的变量。`select()` 允许你基于变量的名称，通过操作快速缩小到一个有用的子集：

- 按名称选择列：

```
flights |>  
  select(year, month, day)
```

- 选择 `year` 和 `day` 之间的所有列：

```
flights |>  
  select(year:day)
```

- 选择除了从 `year` 到 `day` 的所有列：

```
flights |>  
  select(!year:day)
```

历史上，这个操作是用`-`而不是`!`来完成的，所以你可能会在实际场景中看`-`。这两个操作符具有相同的目的，但在行为上有细微的差别。我们推荐使用`!`，因为它读作“不是”，并且与`&`和`|`配合得很好。

- 选择所有是字符的列：

```
flights |>
  select(where(is.character))
```

在 `select()` 中，你可以使用一些辅助函数：

- `starts_with("abc")`: 匹配以 “abc” 开头的名称；
- `ends_with("xyz")`: 匹配以 “xyz” 结尾的名称；
- `contains("ijk")`: 匹配包含 “ijk” 的名称；
- `num_range("x", 1:3)`: 匹配 x1、x2 和 x3。

参阅 `?select` 以获取更多详细信息。一旦你熟悉了正则表达式（这是章节 ?? 的内容），你也可以使用 `matches()` 来选择匹配某个模式的变量。

你可以在 `select()` 变量时使用 `=` 来重命名它们，新名称出现在 `=` 的左侧，旧变量出现在右侧：

```
flights |>
  select(tail_num = tailnum)
#> # A tibble: 336,776 x 1
#>   tail_num
#>   <chr>
#> 1 N14228
#> 2 N24211
#> 3 N619AA
#> 4 N804JB
#> 5 N668DN
#> 6 N39463
#> # i 336,770 more rows
```

### 3 数据转换

#### 3.3.3 rename()

如果你想保留所有现有的变量，只是对几个重命名，可以使用 `rename()` 而不是 `select()`:

```
flights |>
  rename(tail_num = tailnum)
#> # 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     1      517          515       2     830          812
#> 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          830
#> 6 2013     1     1      554          558      -4     740          720
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

如果您有一堆命名不一致的列，并且手工修复它们会很痛苦，请查看 `jani-tor::clean_names()`，它提供了一些有用的自动清理方法。

#### 3.3.4 relocate()

使用 `relocate()` 来移动变量。你可能希望将相关变量收集在一起或将重要变量移到前面。默认情况下，`relocate()` 将变量移到前面:

```
flights |>
  relocate(time_hour, air_time)
#> # A tibble: 336,776 x 19
#>   time_hour           air_time   year month   day dep_time sched_dep_time
#>   <dttm>             <dbl> <int> <int> <int>    <int>       <int>
#> 1 2013-01-01 05:00:00     227  2013     1     1      517        515
#> 2 2013-01-01 05:00:00     227  2013     1     1      533        529
#> 3 2013-01-01 05:00:00     160  2013     1     1      542        540
#> 4 2013-01-01 05:00:00     183  2013     1     1      544        545
#> 5 2013-01-01 06:00:00     116  2013     1     1      554        600
#> 6 2013-01-01 05:00:00     150  2013     1     1      554        558
#> # i 336,770 more rows
#> # i 12 more variables: dep_delay <dbl>, arr_time <int>, ...
```

你也可以使用`.before` 和`.after` 参数指定放置它们的位置，就像`mutate()`一样：

```
flights |>
  relocate(year:dep_time, .after = time_hour)
flights |>
  relocate(starts_with("arr"), .before = dep_time)
```

### 3.3.5 练习

1. 比较`dep_time`、`sched_dep_time` 和`dep_delay`，你认为这三个数字是如何联系起来的？

### 3 数据转换

2. 头脑风暴，想尽可能多的从 flights 中选择变量 dep\_time、dep\_delay、arr\_time 和 arr\_delay 的方法。
3. 如果在 select() 调用中多次指定相同变量的名称，会出现什么情况？
4. any\_of() 函数的作用是什么？为什么它和下面这个向量结合会有用？

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

5. 运行以下代码的结果是否让你感到惊讶？默认情况下，select() 的辅助函数如何处理大写和小写？如何更改该默认值？

```
flights |> select(contains("TIME"))
```

6. 将 air\_time 重命名为 air\_time\_min，以表明度量单位，并将其移动到数据框的开头。
7. 为什么下面的代码不运行，error 代表什么意思？

```
flights |>
  select(tailnum) |>
  arrange(arr_delay)
#> Error in `arrange()`:
#> i In argument: `..1 = arr_delay`.
#> Caused by error:
#> ! object 'arr_delay' not found
```

## 3.4 管道

上面已经展示了管道的简单示例，但是当你开始组合使用多个函数时，它的真正功能才会出现。例如，假设你想查找飞往休斯顿 IAH 机场的最快航班，您需

### 3.4 管道

要组合函数 filter()、mutate()、select() 和 arrange():

```
flights |>
  filter(dest == "IAH") |>
  mutate(speed = distance / air_time * 60) |>
  select(year:day, dep_time, carrier, flight, speed) |>
  arrange(desc(speed))

#> # A tibble: 7,198 x 7
#>   year month   day dep_time carrier flight   speed
#>   <int> <int> <int>    <int> <chr>   <int>   <dbl>
#> 1 2013     7     9      707  UA       226   522.
#> 2 2013     8     27     1850  UA      1128   521.
#> 3 2013     8     28      902  UA      1711   519.
#> 4 2013     8     28     2122  UA      1022   519.
#> 5 2013     6     11     1628  UA      1178   515.
#> 6 2013     8     27     1017  UA       333   515.
#> # i 7,192 more rows
```

即使这个管道有四个步骤，也很容易浏览，因为每个步骤的函数都位于每行的开头：从 flights 数据开始，然后过滤，转换，选择，最后排序。

如果我们不使用管道会发生什么？我们可以将每个函数调用嵌套在前一个调用内部：

```
arrange(
  select(
    mutate(
      filter(
```

### 3 数据转换

```
flights,  
dest == "IAH"  
)  
speed = distance / air_time * 60  
,  
year:day, dep_time, carrier, flight, speed  
,  
desc(speed)  
)
```

或者可以使用一些中间对象：

```
flights1 <- filter(flights, dest == "IAH")  
flights2 <- mutate(flights1, speed = distance / air_time * 60)  
flights3 <- select(flights2, year:day, dep_time, carrier, flight, speed)  
arrange(flights3, desc(speed))
```

虽然这两种形式都有其适用的时间和场合，但管道通常会产生更易于编写和阅读的数据分析代码。

要在代码中添加管道，我们建议使用内置的键盘快捷键 Ctrl/Cmd + Shift + M。要使用 |> 而不是%>%，你需要对 RStudio 选项进行一项更改，如图 ?? 所示；稍后会详细介绍%>%。

#### i Magrittr

如果您已经使用了一段时间的 tidyverse，那么你可能对 magrittr 包提供的%>% 管道很熟悉。**m agritr** 包包含在 tidyverse 中，所以你可以在加载 tidyverse 时使用%>%:

### 3.5 分组

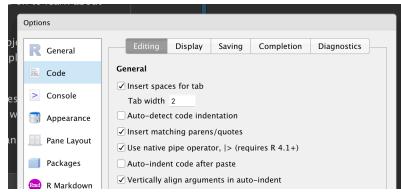


图 3.1: To insert |>, make sure the “Use native pipe operator” option is checked.

```
library(tidyverse)

mtcars %>%
  group_by(cyl) %>%
  summarize(n = n())
```

对于简单的情况，|> 和%>% 的行为是相同的。那么我们为什么推荐基础管道 |> 呢？首先，因为它是 R 语言的基础部分，所以你总是可以使用它，即使你没有使用 tidyverse。其次，|> 比%>% 简单得多：在%>% 于 2014 年被发明和 |> 于 2021 年在 R 4.1.0 中被引入之间的这段时间里，我们对管道有了更好的理解。这使得 |> 能够舍弃那些不常用且不太重要的特性。

## 3.5 分组

到目前为止，你已经学习了处理行和列的函数。当你添加了处理分组的能力时，dplyr 会变得更加强大。在本节中，我们将重点关注最重要的函数：group\_by()、summarize() 和 slice 函数族。

### 3 数据转换

#### 3.5.1 group\_by()

使用 group\_by() 将数据集划分为对分析有意义的组:

```
flights |>
  group_by(month)
#> # A tibble: 336,776 x 19
#> # Groups:   month [12]
#>   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          813
#> 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         1023
#> 5 2013     1     1      554          600      -6     812          837
#> 6 2013     1     1      554          558      -4     740          728
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

group\_by() 并不改变数据，但如果你仔细观察输出，你会注意到输出显示它是按月“分组”(Groups: month [12])。这意味着后续的操作将“按月”进行。group\_by() 将这个分组特性（称为类）添加到数据框中，从而改变了后续应用于数据的函数的行为。

### 3.5.2 summarize()

最重要的分组操作是汇总，如果用于计算单个汇总统计量，它将数据框缩减为每个组只有一行。在 dplyr 中，这个操作是通过 `summarize()`<sup>3</sup> 完成的，如下例所示，它计算了每个月的平均离港延误时间：

```
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay)
  )
#> # A tibble: 12 x 2
#>   month avg_delay
#>   <int>     <dbl>
#> 1     1        NA
#> 2     2        NA
#> 3     3        NA
#> 4     4        NA
#> 5     5        NA
#> 6     6        NA
#> # i 6 more rows
```

哎呀！出错了，我们所有的结果都是 NA（读作“N-A”），这是 R 中缺失值的符号。这是因为我们观察的一些航班在延误列中有缺失数据，所以当我们将这些值包括在内计算平均值时，我们就得到了一个 NA 结果。我们将在章节 ?? 部分中详细讨论缺失值，但现在我们会告诉 `mean()` 函数通过设置 `na.rm` 参数为 `TRUE` 来忽略所有缺失值：

---

<sup>3</sup> 如果你喜欢英式英语，也可以用 `summarise()`。

### 3 数据转换

```
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE)
  )
#> # A tibble: 12 x 2
#>   month  avg_delay
#>   <int>     <dbl>
#> 1     1      10.0
#> 2     2      10.8
#> 3     3      13.2
#> 4     4      13.9
#> 5     5      13.0
#> 6     6      20.8
#> # i 6 more rows
```

你可以在单次调用 `summarize()` 时创建任意数量的汇总。在接下来的章节中，你将学习各种有用的汇总，但其中一个非常有用的摘要是 `n()`，它返回每个组的行数

```
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  )
#> # A tibble: 12 x 3
#>   month  avg_delay     n
#>   <int>     <dbl> <int>
```

```
#>   <int>    <dbl> <int>
#> 1     1      10.0 27004
#> 2     2      10.8 24951
#> 3     3      13.2 28834
#> 4     4      13.9 28330
#> 5     5      13.0 28796
#> 6     6      20.8 28243
#> # i 6 more rows
```

Means 和 counts 在数据科学中可以让你走得很远，远到让你吃惊！

### 3.5.3 slice\_ 函数族

有五个函数允许你方便地提取每个组中内特定行：

- `df |> slice_head(n = 1)` 从每个组中取第一行；
- `df |> slice_tail(n = 1)` 从每个组中取最后一行；
- `df |> slice_min(x, n = 1)` 取列 `x` 值最小的行；
- `df |> slice_max(x, n = 1)` 取列 `x` 值最大的行；
- `df |> slice_sample(n = 1)` 随机取一行。

你可以改变参数 `n` 来选择多行，或者代替 `n=`，你可以使用 `prop = 0.1`（例如）来选择每个组的 10% 的行。例如，以下代码查找每个目的地到达时延误最严重的航班：

```
flights |>
  group_by(dest) |>
  slice_max(arr_delay, n = 1) |>
```

### 3 数据转换

```
relocate(dest)

#> # A tibble: 108 x 19
#> # Groups:   dest [105]
#>   dest    year month   day dep_time sched_dep_time dep_delay arr_time
#>   <chr> <int> <int> <int>     <int>          <int>      <dbl>     <int>
#> 1 ABQ    2013     7     22     2145        2007       98     132
#> 2 ACK    2013     7     23     1139       800      219    1250
#> 3 ALB    2013     1     25     123        2000      323     229
#> 4 ANC    2013     8     17     1740       1625       75    2042
#> 5 ATL    2013     7     22     2257       759      898     121
#> 6 AUS    2013     7     10     2056       1505      351    2347
#> # i 102 more rows
#> # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, ...
```

请注意，虽然有 105 个目的地，但我们这里得到了 108 行，这是怎么回事？slice\_min() 和 slice\_max() 会保留并列的值，所以 n = 1 意味着给我们所有具有最高值的行。如果你想要每个组正好一行，你可以设置 with\_ties = FALSE。

这与使用 summarize() 计算最大延误类似，但你会得到整个对应的行（如果有并列值，则是多行），而不是单个汇总统计量。

#### 3.5.4 按多变量分组

可以使用多个变量创建分组。例如，我们可以为每个日期创建一个组。

You can create groups using more than one variable. For example, we could make a group for each date.

### 3.5 分组

```
daily <- flights |>
  group_by(year, month, day)
daily
#> # A tibble: 336,776 x 19
#> # Groups:   year, month, day [365]
#>   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
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

当你在使用多个变量对 tibble 进行分组并对其进行汇总时，每个汇总都会去掉最后一个分组。事后看来，这不是让这个函数工作的好方法，但如果不行破坏现有代码就很难改变它。为了让发生的事情变得显而易见，dplyr 显示了一条消息，告诉你如何改变这种行为：

```
daily_flights <- daily |>
  summarise(n = n())
#> `summarise()` has grouped output by 'year', 'month'. You can override using
#> the ` `.groups` argument.
```

如果你对这种行为感到满意，你可以明确地要求它以抑制这条消息：

### 3 数据转换

```
daily_flights <- daily |>  
  summarize(  
    n = n(),  
    .groups = "drop_last"  
)
```

或者，通过设置不同的值来改变默认行为，例如，“drop” 删除所有分组，或者“keep”保留相同的分组。

#### 3.5.5 去消分组

您可能还希望在不使用 `summarize()` 的情况下从数据框中删除分组。您可以使用 `ungroup()` 实现这一点。

```
daily |>  
  ungroup()  
#> # 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     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          830  
#> 6 2013     1     1      554          558       -4     740          720  
#> # i 336,770 more rows  
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

### 3.5 分组

现在让我们看看在汇总未分组的数据框时会发生什么。

```
daily |>
ungroup() |>
summarize(
  avg_delay = mean(dep_delay, na.rm = TRUE),
  flights = n()
)
#> # A tibble: 1 x 2
#>   avg_delay flights
#>       <dbl>    <int>
#> 1      12.6  336776
```

返回一行，因为 dplyr 将未分组数据框中的所有行视为属于一个组。

### 3.5.6 .by

dplyr 1.1.0 引入了一个新的处于试验阶段的语法，用于每次操作的分组，即.by 参数。group\_by() 和 ungroup() 不会被弃用，但现在你也可以在单个操作中使用.by 参数进行分组：

```
flights |>
summarize(
  delay = mean(dep_delay, na.rm = TRUE),
  n = n(),
  .by = month
)
```

### 3 数据转换

或者如果你想按多个变量分组:

```
flights |>
  summarize(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .by = c(origin, dest)
  )
```

.by 与所有动词（函数）兼容，其优势在于你不需要使用.groups 参数来抑制分组消息，或在完成后使用 ungroup()。

我们在编写本书时，没有重点关注这种语法，因为它当时非常新。但我们还是想提一下，因为我们认为它很有前景，并且可能会非常受欢迎。你可以在 [dplyr 1.1.0 的博客](#) 中了解更多关于它的信息。

#### 3.5.7 练习

1. 哪家航空公司的平均延误时间最长？挑战：你能理清机场差和航空公司差的影响吗？为什么能/为什么不能？（提示：考虑使用 flights |> group\_by(carrier, dest) |> summarize(n())）
2. 找出从每个目的地出发延误最严重的航班；
3. 延误情况在一天中是如何变化的？用图表来展示你的答案；
4. 如果你给 slice\_min() 及其相关函数提供负的 n 值会怎样？
5. 用你刚刚学习的 dplyr 动词（函数）来解释 count() 做了什么。count() 的 sort 参数有什么作用？

### 3.5 分组

6. 假设我们有以下的小数据框:

```
df <- tibble(  
  x = 1:5,  
  y = c("a", "b", "a", "a", "b"),  
  z = c("K", "K", "L", "L", "K")  
)
```

- a. 写下你认为的输出是什么样子，然后检查是否正确，并描述 group\_by() 的作用。

```
df |>  
  group_by(y)
```

- b. 写下你认为的输出是什么样子，然后检查你是否正确，并描述 arrange() 的作用。同时评论一下它与 (a) 部分中的 group\_by() 有什么不同。

```
df |>  
  arrange(y)
```

- c. 写下你认为的输出是什么样子，然后检查你是否正确，并描述管道的功能。

```
df |>  
  group_by(y) |>  
  summarize(mean_x = mean(x))
```

- d. 写下你认为的输出是什么样子，然后检查你是否正确，并描述管道的功能。然后对信息内容进行评论。

### 3 数据转换

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

- e. 写下你认为的输出是什么样子，然后检查你是否正确，并描述管道的功能。与 (d) 部分中的输出有何不同？

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x), .groups = "drop")
```

- f. 写下你认为的输出是什么样子，然后检查你是否正确，并描述每个管道的作用。两个管道的输出有什么不同？

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))

df |>
  group_by(y, z) |>
  mutate(mean_x = mean(x))
```

## 3.6 案例研究：汇总数据和样本量

无论何时进行数据汇总，包含计数 (`n()`) 总是一个好主意。这样可以确保你不是基于很少量的数据来得出结论。我们将使用 **Lahman** 包中的棒球数据来演示这一点。具体来说，我们将比较球员击球成功 (H) 的次数与尝试击球 (AB) 的次数之间的比例：

### 3.6 案例研究：汇总数据和样本量

```
batters <- Lahman::Batting |>
  group_by(playerID) |>
  summarize(
    performance = sum(H, na.rm = TRUE) / sum(AB, na.rm = TRUE),
    n = sum(AB, na.rm = TRUE)
  )
batters
#> # A tibble: 20,469 x 3
#>   playerID  performance     n
#>   <chr>        <dbl> <int>
#> 1 aardsda01      0         4
#> 2 aaronha01    0.305    12364
#> 3 aaronto01    0.229     944
#> 4 aasedo01      0         5
#> 5 abadan01     0.0952    21
#> 6 abadfe01     0.111     9
#> # i 20,463 more rows
```

当我们绘制击球手的技巧（通过击球率来衡量，即 `performance`）与击球机会的数量（通过击球次数来衡量，即 `n`）之间的关系时，你会看到两种模式：

1. 击球次数较少的球员之间的技巧差异更大。这种图的形状非常典型：每当你绘制平均值（或其他汇总统计量）与组大小时，你会看到随着样本量的增加，变异程度会降低<sup>4</sup>。
2. 技巧（`performance`）与击球机会（`n`）之间存在正相关关系，因为球队希望给他们的最佳击球手最多的击球机会。

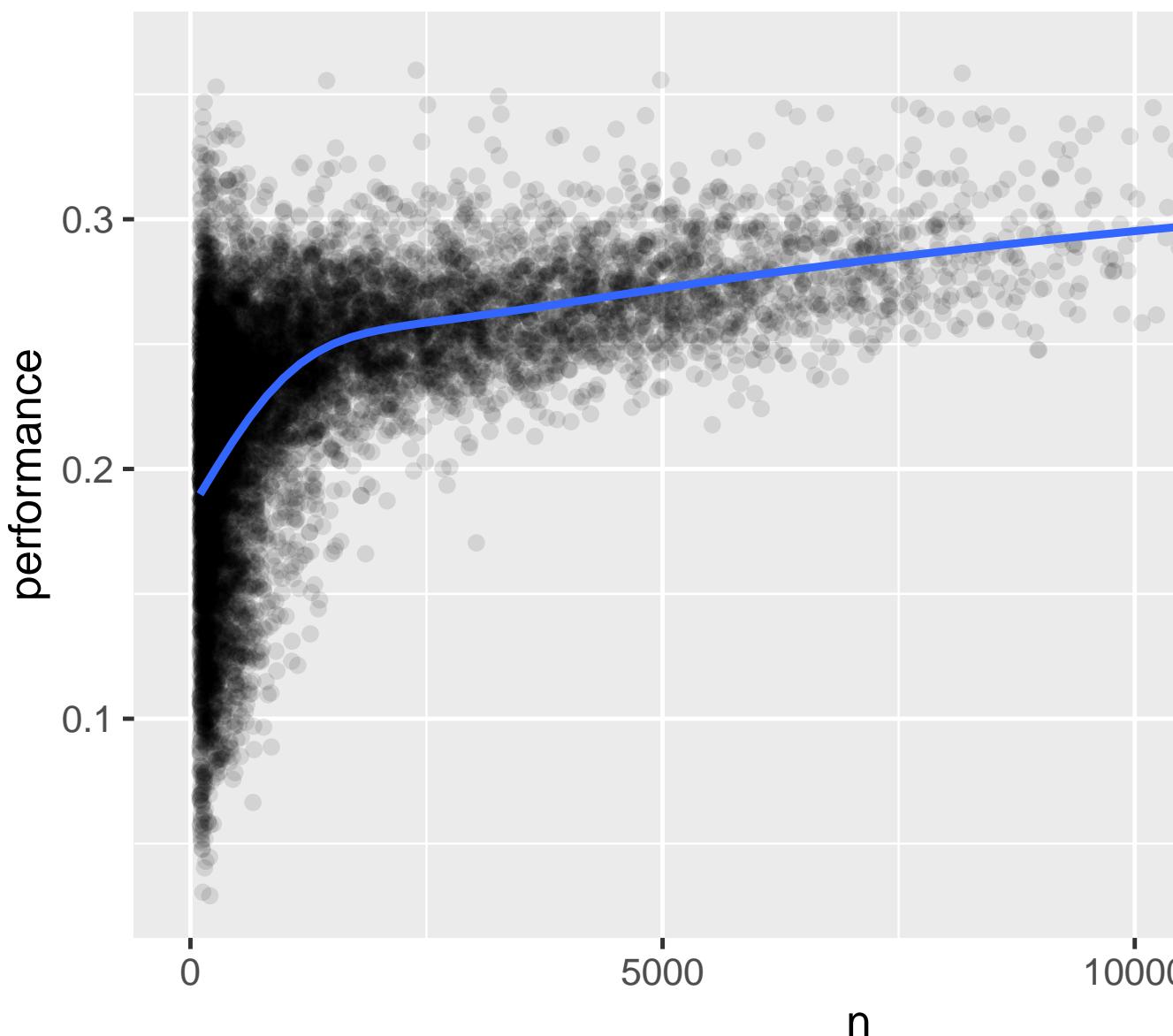
---

<sup>4</sup>大数定律

### 3 数据转换

```
batters |>
  filter(n > 100) |>
  ggplot(aes(x = n, y = performance)) +
  geom_point(alpha = 1 / 10) +
  geom_smooth(se = FALSE)
```

### 3.6 案例研究：汇总数据和样本量



### 3 数据转换

请注意将 `ggplot2` 和 `dplyr` 结合使用的便捷模式。你只需要记住在数据集处理时使用 `|>`，而在向你的图添加图层时使用 `+`。

这对于排名也有重要的影响。如果你只是简单地根据 `desc(performance)` 进行排序，那么显然击球率最好的人是那些尝试击球次数很少但碰巧击中的人，他们不一定是技术最熟练的球员：

```
batters |>
  arrange(desc(performance))
#> # A tibble: 20,469 x 3
#>   playerID  performance     n
#>   <chr>      <dbl> <int>
#> 1 abramge01      1     1
#> 2 alberan01      1     1
#> 3 banisje01      1     1
#> 4 bartocl01      1     1
#> 5 bassdo01      1     1
#> 6 birasst01      1     2
#> # i 20,463 more rows
```

关于这个问题及其解决方法，你可以在以下网址找到很好的解释：[http://varianceexplained.org/r/empirical\\_bayes\\_baseball/](http://varianceexplained.org/r/empirical_bayes_baseball/) 和 <https://www.evanmiller.org/how-not-to-sort-by-average-rating.html>.

## 3.7 小结

在本章中，你学习了 `dplyr` 为处理数据框提供的工具。这些工具大致分为三类：操作行的工具（如 `filter()` 和 `arrange()`），操作列的工具（如 `select()`

### 3.7 小结

和 `mutate()`，以及操作组的工具（如 `group_by()` 和 `summarize()`）。在本章中，我们专注于这些“整个数据框”的工具，但你还没有深入了解可以使用单个变量做什么。我们将在本书的转换部分回到这个问题，其中每一章都将为你提供特定类型变量的工具。

在下一章中，我们将回到工作流程，讨论代码风格的重要性，保持您的代码组织良好，以便你和他人能够轻松阅读和理解你的代码。



## 4 工作流程：代码风格

良好的编码风格就像正确的标点符号：没有它也能写代码，但它确实能让事情变得更易于阅读。即使作为一名非常新的程序员，养成好的代码风格也是一个好主意。使用一致的代码风格可以让他（包括未来的你）更容易读懂你的工作，这在你需要他人帮助时变得尤为重要。本章将介绍[tidyverse 风格指南](#)的最重要内容，该指南在本书中贯穿始终。

刚开始为代码设置样式可能会觉得有些乏味，但如果你坚持练习，它很快就会成为你的第二天性。此外，还有一些很棒的工具可以快速重新格式化现有的代码，比如 Lorenz Walthert 的[styler](#)包。一旦你使用`install.packages("styler")`安装了这个包，一个简单的使用方法是通过RStudio 的命令面板。命令面板允许你使用任何内置的 RStudio 命令以及由包提供的许多插件。通过按 Cmd/Ctrl + Shift + P 打开面板，然后输入“styler”来查看 styler 提供的所有快捷键，图 ?? 展示了结果。

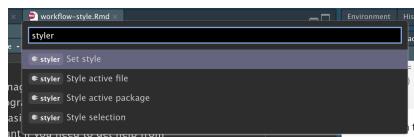


图 4.1: RStudio’s command palette makes it easy to access every RStudio command using only the keyboard.

## 4 工作流程: 代码风格

在本章中，我们将使用 tidyverse 和 nycflights13 包作为代码示例。

```
library(tidyverse)
library(nycflights13)
```

### 4.1 名称

我们在 @sec-whats-in-a-name 简要地讨论了名称。请记住，变量名（通过 `<-` 和 `mutate()` 创建的）应该只使用小写字母、数字和 `_`。使用 `_` 来分隔名称中的单词。

```
# Strive for:
short_flights <- flights |> filter(air_time < 60)

# Avoid:
SHORTFLIGHTS <- flights |> filter(air_time < 60)
```

作为一条通用的经验法则，最好选择长而描述性强的名称，这样更容易理解，而不是快速输入的简洁名称。在编写代码时，短名称节省的时间相对较少（特别是当自动补全功能帮助你完成输入时），但当你回到旧代码并被迫猜测一个晦涩的缩写时，这可能会很耗时。

如果你有一组与相关事物相关的名称，请尽量保持一致。当你忘记之前的约定时，不一致性很容易产生，所以如果你必须回去重命名事物，也不要感到难过。一般来说，如果你有一组变量是某个主题的变体，最好给它们一个共同的前缀而不是共同的后缀，因为自动补全在变量的开头效果最好。

## 4.2 空格

在除 `^` 以外的数学运算符 (即`-`、`==`、`<`...) 以及赋值运算符 (`<-`) 的两侧加上空格。

```
# Strive for
z <- (a + b)^2 / d

# Avoid
z<-( a + b ) ^ 2/d
```

对于常规函数调用，不要在圆括号内或圆括号外放空格。总是在逗号后面加一个空格，就像在标准英语中一样。

```
# Strive for
mean(x, na.rm = TRUE)

# Avoid
mean (x ,na.rm=TRUE)
```

如果可以提高对齐效果，可以添加额外的空格。例如，如果要在 `mutate()` 中创建多个变量，则可能需要添加空格，以便所有的 `=` 对齐<sup>1</sup>。这会使得浏览代码更容易。

---

<sup>1</sup>由于 `dep_time` 是 HMM 或 HHMM 格式，因此我们使用整数除法 (`%/%`) 来获得小时，使用余数 (也称为取模，`%%`) 来获得分钟。

#### 4 工作流程: 代码风格

```
flights |>
  mutate(
    speed      = distance / air_time,
    dep_hour   = dep_time %/% 100,
    dep_minute = dep_time %% 100
  )
```

### 4.3 管道

|> 前面应该始终有一个空格，并且通常应该是行的最后一个元素。这样做可以更容易地添加新步骤、重新排列现有步骤、修改步骤中的元素，并通过浏览左侧的动词（函数）来获得一个全局视角。

```
# Strive for
flights |>
  filter(!is.na(arr_delay), !is.na(tailnum)) |>
  count(dest)

# Avoid
flights|>filter(!is.na(arr_delay), !is.na(tailnum))|>count(dest)
```

如果你正在使用管道传递数据的函数具有被命名的参数（如 `mutate()` 或 `summarize()`），请将每个参数放在新行上。如果函数没有命名参数（如 `select()` 或 `filter()`），则除非参数过长无法放在一行中，否则将所有内容放在一行上。在参数过长的情况下，你应该将每个参数放在自己的行上。

### 4.3 管道

```
# Strive for
flights |>
  group_by(tailnum) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()
  )

# Avoid
flights |>
  group_by(
    tailnum
  ) |>
  summarize(delay = mean(arr_delay, na.rm = TRUE), n = n())
```

在管道的第一步之后，将每行缩进两个空格。在 `|>` 后面的换行后，RStudio 会自动为你添加空格。如果你将每个参数放在单独的一行上，则再缩进两个空格。确保）单独放在一行上，并且不缩进，以匹配函数名的水平位置。

```
# Strive for
flights |>
  group_by(tailnum) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()
  )

# Avoid
```

#### 4 工作流程: 代码风格

```
flights|>
  group_by(tailnum) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()
  )

# Avoid
flights|>
  group_by(tailnum) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()
  )
```

如果你的管道可以很容易地放在一行上，那么忽略一些规则是可以的。但根据我们的共同经验，简短的代码片段经常会变得更长，所以通常一开始就使用所有你需要的垂直空间会在长远上节省时间。

```
# This fits compactly on one line
df |> mutate(y = x + 1)

# While this takes up 4x as many lines, it's easily extended to
# more variables and more steps in the future
df |>
  mutate(
    y = x + 1
  )
```

最后，要警惕编写非常长的管道，比如超过 10-15 行的管道。尝试将它们分解成更小的子任务，并为每个任务赋予一个信息性强的名称。这些名称将帮助读者了解正在发生的事情，并更容易地检查中间结果是否符合预期。只要你能为某个事物赋予一个信息性强的名称，你就应该这样做，例如当你从根本上改变数据的结构时，例如在透视或汇总之后。不要期望第一次就能做对！这意味着，如果中间状态可以得到好的名称，就应该拆分长的管道。

## 4.4 ggplot2

适用于管道的基本规则也适用于 ggplot2，对待 `+` 就像对待 `|>` 一样。

```
flights |>
  group_by(month) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE)
  ) |>
  ggplot(aes(x = month, y = delay)) +
  geom_point() +
  geom_line()
```

同样，如果你不能将函数的所有参数放在一行中，请将每个参数放在单独的一行中：

```
flights |>
  group_by(dest) |>
  summarize(
    distance = mean(distance),
```

## 4 工作流程: 代码风格

```
speed = mean(distance / air_time, na.rm = TRUE)
) |>
ggplot(aes(x = distance, y = speed)) +
geom_smooth(
  method = "loess",
  span = 0.5,
  se = FALSE,
  color = "white",
  linewidth = 4
) +
geom_point()
```

注意从 `|>` 到 `+` 的转换。我们希望这种转换是不必要的，但不幸的是，`ggplot2` 是在管道发明之前编写的。

## 4.5 分段注释

随着你的脚本变长，你可以使用分段注释（sectioning comments）将文件分解成可管理的片段：

```
# Load data -----
# Plot data -----
```

RStudio 提供了一个快捷键来创建这些标题 (Cmd/Ctrl Shift R)，并将它们显示在编辑器左下角的代码导航下拉菜单中，如图 ?? 所示。

## 4.6 练习



图 4.2: After adding sectioning comments to your script, you can easily navigate to them using the code navigation tool in the bottom-left of the script editor.

## 4.6 练习

1. 按照上面的指南重新设计以下管道的样式。

```
flights|>filter(dest=="IAH")|>group_by(year,month,day)|>summarize(n=n(),
delay=mean(arr_delay,na.rm=TRUE))|>filter(n>10)

flights|>filter(carrier=="UA",dest%in%c("IAH","HOU"),sched_dep_time>
0900,sched_arr_time<2000)|>group_by(flight)|>summarize(delay=mean(
arr_delay,na.rm=TRUE),cancelled=sum(is.na(arr_delay)),n=n())|>filter(n>10)
```

## 4.7 小结

在本章中，你学习了代码风格最重要的原则。这些原则一开始可能感觉像是一套任意的规则（因为它们确实是），但随着时间的推移，当你编写更多的代码并与更多的人分享代码时，你会看到一致的样式是多么重要。别忘了 `styler` 包，它是快速提高样式不佳的代码质量的绝佳方式。

在下一章中，我们将回到数据科学工具，学习整理数据。整理数据是一种组织数据框的一致方式，它在 `tidyverse` 中得到了广泛应用。这种一致性让你的生活变得更加容易，因为一旦你有了整洁的数据，它就可以与绝大多数 `tidyverse`

#### 4 工作流程: 代码风格

函数一起工作。当然，生活从来都不容易，你在实际中遇到的大多数数据集都不会是整洁的。所以，我们还将教你如何使用 `tidyverse` 包来整理你的不整洁的数据。

# 5 数据整理

## 5.1 引言

“幸福的家庭都是相似的；每个不幸的家庭各有各的不幸。”

—列夫·托尔斯泰

“整洁的数据集都是相似的，但每个混乱的数据集各有各的混乱方式。”

—哈德利·威克汉姆

在本章中，你将学习一种在 R 中组织数据的一致方法，即利用一种被称为整齐数据（tidy data）的系统。将数据整理成这种格式需要一些前期工作，但从长远来看，这些工作是值得的。一旦你有了整齐的数据和 tidyverse 包提供的整理工具，你将花费更少的时间将数据从一种表现形式转换为另一种表现形式，从而让你有更多的时间投入到你关心的数据问题上。

在本章中，你将首先学习整齐数据的定义，并将其应用于一个简单的示例数据集。然后，我们将深入探讨用于整理数据的主要工具：数据重塑（data pivoting），数据透视允许你改变数据的格式而不改变任何值。

## 5 数据整理

### 5.1.1 必要条件

本章将重点介绍 `tidyverse`，它提供了一系列工具来帮助整理混乱的数据集，是 `tidyverse` 的一个成员。

```
library(tidyverse)
```

从本章开始，我们将阻止 `library(tidyverse)` 的加载消息。

## 5.2 整齐数据

你可以用多种方式表示相同的基础数据。下面的例子展示了以三种不同的方式组织的同一数据。每个数据集都显示了四个变量的相同值：国家（*country*）、年份（*year*）、人口（*population*）和记录在案的结核病（tuberculosis, TB）病例数（*cases*），但每个数据集都以不同的方式组织这些值。

```
table1
#> # A tibble: 6 x 4
#>   country     year   cases population
#>   <chr>       <dbl>   <dbl>      <dbl>
#> 1 Afghanistan 1999     745 19987071
#> 2 Afghanistan 2000    2666 20595360
#> 3 Brazil       1999  37737 172006362
#> 4 Brazil       2000  80488 174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

```
table2
#> # A tibble: 12 x 4
#>   country     year type    count
#>   <chr>       <dbl> <chr>    <dbl>
#> 1 Afghanistan 1999 cases     745
#> 2 Afghanistan 1999 population 19987071
#> 3 Afghanistan 2000 cases     2666
#> 4 Afghanistan 2000 population 20595360
#> 5 Brazil      1999 cases     37737
#> 6 Brazil      1999 population 172006362
#> # i 6 more rows
```

```
table3
#> # A tibble: 6 x 3
#>   country     year rate
#>   <chr>       <dbl> <chr>
#> 1 Afghanistan 1999 745/19987071
#> 2 Afghanistan 2000 2666/20595360
#> 3 Brazil      1999 37737/172006362
#> 4 Brazil      2000 80488/174504898
#> 5 China       1999 212258/1272915272
#> 6 China       2000 213766/1280428583
```

它们都表示了同一基础数据，但使用难度并不相同。其中，`table1` 在 `tidyverse` 中更容易使用，因为它很整齐（tidy）。

使一个数据集整齐的三个相互关联的规则是：

## 5 数据整理

1. 每个变量都是一列；每一列都是一个变量；
2. 每个观测值都是一行；每一行都是一个观测值；
3. 每个值都是一个单元格；每个单元格都是一个单一的值。

图 ?? 直观地展示了这些规则。

country	year	cases	population
Afghanistan	1999	45	1637071
Afghanistan	2000	6666	20595360
Brazil	1999	3737	17206362
Brazil	2000	80488	17404898
China	1999	212258	1272515272
China	2000	21666	128048583

variables

country	year	cases	population
Afghanistan	1999	45	1637071
Afghanistan	2000	6666	20595360
Brazil	1999	3737	17206362
Brazil	2000	80488	17404898
China	1999	212258	1272515272
China	2000	21666	128048583

observations

country	year	cases	population
Afghanistan	1999	45	1637071
Afghanistan	2000	6666	20595360
Brazil	1999	3737	17206362
Brazil	2000	80488	17404898
China	1999	212258	1272515272
China	2000	21666	128048583

values

图 5.1: The following three rules make a dataset tidy: variables are columns, observations are rows, and values are cells.

为什么要确保你的数据是整齐的？有两个主要优势：

1. 选择一种一致的数据存储方式具有普遍优势。如果你的数据结构是一致的，学习与之配合使用的工具就更容易，因为它们具有底层的一致性。
2. 将变量放在列中具有特定的优势，因为这可以让 R 的矢量化特性大放异彩。正如你在 @sec-mutate 和 @sec-summarize 所学到的，大多数内置的 R 函数都使用值的向量。这使得处理整齐数据感觉特别自然。

dplyr、ggplot2 以及 tidyverse 中的所有其他包都是为处理整齐数据而设计的。下面代码是一些展示如何使用 `table1` 的小示例。

## 5.2 整齐数据

```
# Compute rate per 10,000
table1 |>
  mutate(rate = cases / population * 10000)
#> # A tibble: 6 x 5
#>   country     year   cases population   rate
#>   <chr>      <dbl>   <dbl>       <dbl>   <dbl>
#> 1 Afghanistan 1999     745  19987071 0.373
#> 2 Afghanistan 2000    2666  20595360 1.29
#> 3 Brazil      1999  37737  172006362 2.19
#> 4 Brazil      2000  80488  174504898 4.61
#> 5 China       1999 212258 1272915272 1.67
#> 6 China       2000 213766 1280428583 1.67

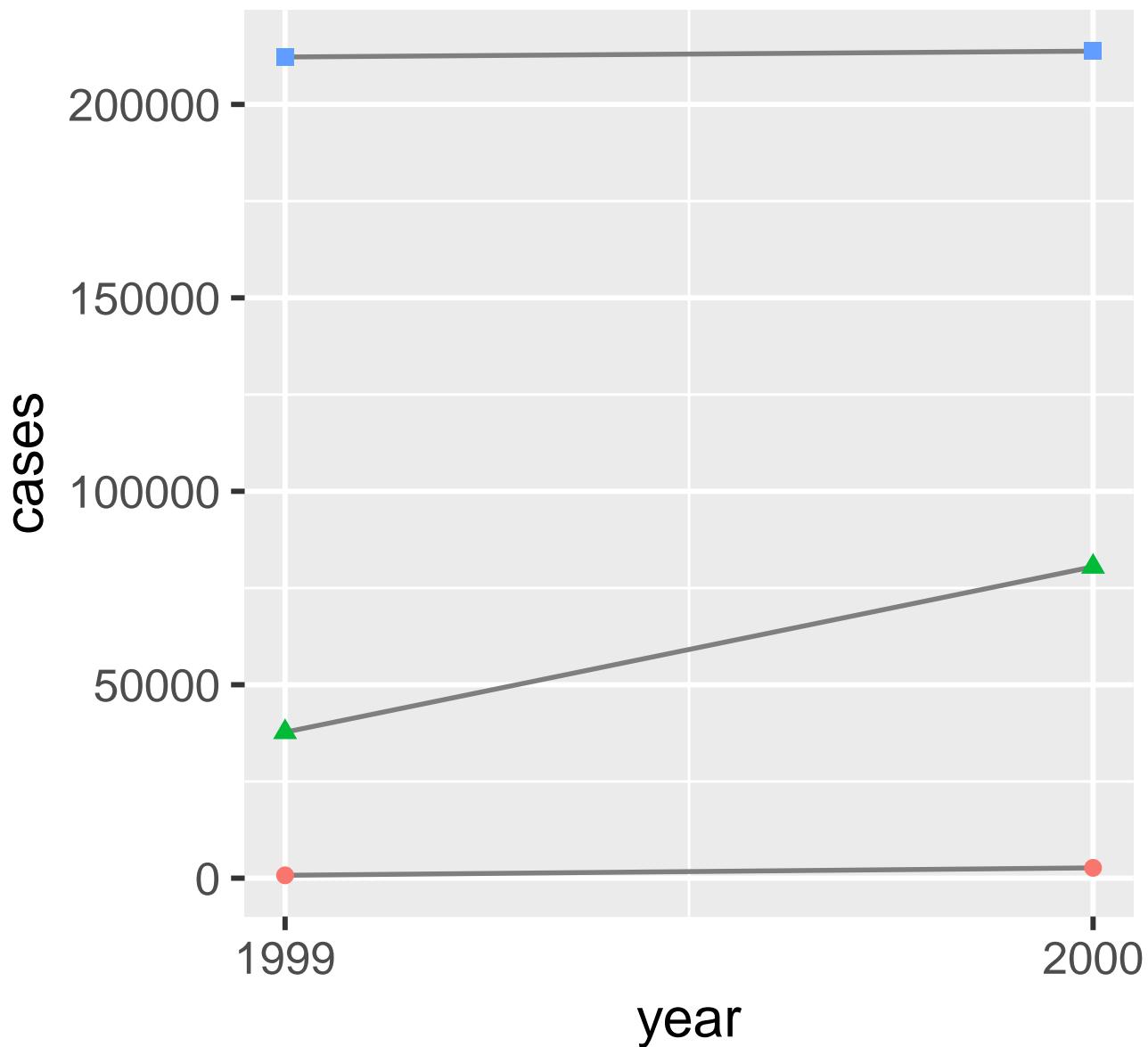
# Compute total cases per year
table1 |>
  group_by(year) |>
  summarize(total_cases = sum(cases))
#> # A tibble: 2 x 2
#>   year total_cases
#>   <dbl>      <dbl>
#> 1 1999      250740
#> 2 2000      296920

# Visualize changes over time
ggplot(table1, aes(x = year, y = cases)) +
  geom_line(aes(group = country), color = "grey50") +
  geom_point(aes(color = country, shape = country)) +
```

## 5 数据整理

```
scale_x_continuous(breaks = c(1999, 2000)) # x-axis breaks at 1999 and 2000
```

## 5.2 整齐数据



### 5.2.1 练习

1. 对于每个示例表格，描述每个观测值和每列分别代表什么。
2. 概述你将如何计算 `table2` 和 `table3` 中的 `rate`。你需要执行四个操作：
  - a. 提取每个国家每年的 TB 病例数；
  - b. 提取每个国家每年对应的人口数；
  - c. 将病例数除以人口数，然后乘以 10000；
  - d. 将结果存储回适当的地方。

你还没有学习所有实际执行这些操作所需的函数，但应该仍然能够思考出所需的转换步骤。

## 5.3 数据转换——长格式

整齐数据的原则可能看起来如此显而易见，以至于你怀疑是否会遇到不整齐的数据集。然而，不幸的是，大多数真实数据都是不整齐的。这主要有两个原因：

1. 数据通常是为了方便除了分析以外的其他目的而组织的。例如，数据通常以易于数据录入而不是分析的结构来组织。
2. 大多数人并不熟悉整齐数据的原则，除非你花很多时间处理数据，否则很难自己推导出这些原则。

这意味着大多数实际分析至少需要进行一些整理。首先要弄清楚基础变量和观测值是什么，有时这很简单，有时你需要咨询最初生成数据的人；接下来，你需要将数据重塑（**pivot**）为整齐的形式，将变量放在列中，将观测值放在行中。

### 5.3 数据转换—长格式

`tidyR` 包提供了两个用于数据重塑的函数：`pivot_longer()` 和 `pivot_wider()`。我们将首先从 `pivot_longer()` 开始，因为它是最常见的情况。让我们深入探讨一些示例。

#### 5.3.1 列名中的数据

`billboard` 数据集记录了 2000 年歌曲的 Billboard 排行榜排名：

```
billboard
#> # A tibble: 317 x 79
#>   artist      track      date.entered    wk1    wk2    wk3    wk4    wk5
#>   <chr>      <chr>      <date>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#> 1 2 Pac      Baby Don't Cry (Ke~ 2000-02-26     87     82     72     77     87
#> 2 2Ge+her    The Hardest Part O~ 2000-09-02     91     87     92     NA     NA
#> 3 3 Doors Down Kryptonite    2000-04-08     81     70     68     67     66
#> 4 3 Doors Down Loser       2000-10-21     76     76     72     69     67
#> 5 504 Boyz    Wobble Wobble    2000-04-15     57     34     25     17     17
#> 6 98^0        Give Me Just One N~ 2000-08-19     51     39     34     26     26
#> # i 311 more rows
#> # i 71 more variables: wk6 <dbl>, wk7 <dbl>, wk8 <dbl>, wk9 <dbl>, ...
```

在这个数据集中，每个观测都是一首歌。前三列（`artist`、`track` 和 `date.entered`）是描述歌曲的变量。然后有 76 列（`wk1-wk76`）描述了歌曲在每周的排名<sup>1</sup>。在这里，列名是一个变量（`week`），而单元格的值是另一个变量（`rank`）。

为了整理这个数据，我们使用 `pivot_longer()`：

---

<sup>1</sup>只要这首歌在 2000 年的某个时间点进入排行榜前 100 名，它就会被收录，并在它出现后的 72 周内被追踪。

## 5 数据整理

```
billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank"
  )
#> # A tibble: 24,092 x 5
#>   artist track           date.entered week   rank
#>   <chr>  <chr>          <date>       <chr> <dbl>
#> 1 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk1     87
#> 2 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk2     82
#> 3 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk3     72
#> 4 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk4     77
#> 5 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk5     87
#> 6 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk6     94
#> 7 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk7     99
#> 8 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk8     NA
#> 9 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk9     NA
#> 10 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk10    NA
#> # i 24,082 more rows
```

数据之后，有三个关键参数：

- `cols` 指定了哪些列需要进行重塑，即哪些列不是变量。这个参数使用了与 `select()` 相同的语法，因此在这里我们可以使用!`c(artist, track, date.entered)` 或者 `starts_with("wk")`。
- `names_to` 用于命名列名中存储的变量，我们将其命名为 `week`
- `values_to` 用于命名单元格值中存储的变量，我们将其命名为 `rank`。

### 5.3 数据转换—长格式

请注意，代码中“`week`”和“`rank`”被引号括起来，因为这些是我们创建的新变量，在运行 `pivot_longer()` 调用时它们还不在数据中。

现在让我们把注意力转向得到的长数据框。如果一首歌在前 100 名中的时间少于 76 周，会发生什么？以 2 Pac 的《Baby Don't Cry》为例。上面的输出表明它只在前 100 名中待了 7 周，而其余所有周都填充了缺失值。这些 NA 并不真正代表未知的观测值，它们是由于数据集的结构而被迫存在的，因此我们可以要求 `pivot_longer()` 通过设置 `values_drop_na = TRUE` 来删除它们：

```
billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  )
#> # A tibble: 5,307 x 5
#>   artist track           date.entered week   rank
#>   <chr>  <chr>          <date>      <chr> <dbl>
#> 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26 wk1     87
#> 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26 wk2     82
#> 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26 wk3     72
#> 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26 wk4     77
#> 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26 wk5     87
#> 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26 wk6     94
#> # i 5,301 more rows
```

现在行数大大减少，这表明许多包含 NA 的行已被删除。

## 5 数据整理

你也可能会好奇，如果一首歌在前 100 名中超过 76 周，会发生什么？从这些数据中我们无法得知，但你可以猜测数据集会添加额外的列，如 `wk77`、`wk78` 等。

现在数据已经整齐了，但我们可以使用 `mutate()` 和 `readr::parse_number()` 将 `week` 的值从字符串转换为数字，以使未来的计算更加方便。`parse_number()` 是一个方便的函数，它会从字符串中提取第一个数字，忽略其他所有文本。

```
billboard_longer <- billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  ) |>
  mutate(
    week = parse_number(week)
  )
billboard_longer
#> # A tibble: 5,307 x 5
#>   artist track           date.entered  week  rank
#>   <chr>  <chr>          <date>        <dbl> <dbl>
#> 1 2 Pac Baby Don't Cry (Keep... 2000-02-26     1     87
#> 2 2 Pac Baby Don't Cry (Keep... 2000-02-26     2     82
#> 3 2 Pac Baby Don't Cry (Keep... 2000-02-26     3     72
#> 4 2 Pac Baby Don't Cry (Keep... 2000-02-26     4     77
#> 5 2 Pac Baby Don't Cry (Keep... 2000-02-26     5     87
#> 6 2 Pac Baby Don't Cry (Keep... 2000-02-26     6     94
#> # i 5,301 more rows
```

### 5.3 数据转换—长格式

既然我们已经将所有周数放入一个变量，并将所有排名值放入另一个变量，现在可以很好地可视化歌曲排名如何随时间变化。下面的代码展示了这一点，结果位于 @fig-billboard-ranks。我们可以看到，很少有歌曲能在前 100 名中保持超过 20 周的时间。

```
billboard_longer |>
  ggplot(aes(x = week, y = rank, group = track)) +
  geom_line(alpha = 0.25) +
  scale_y_reverse()
```

## 5 数据整理

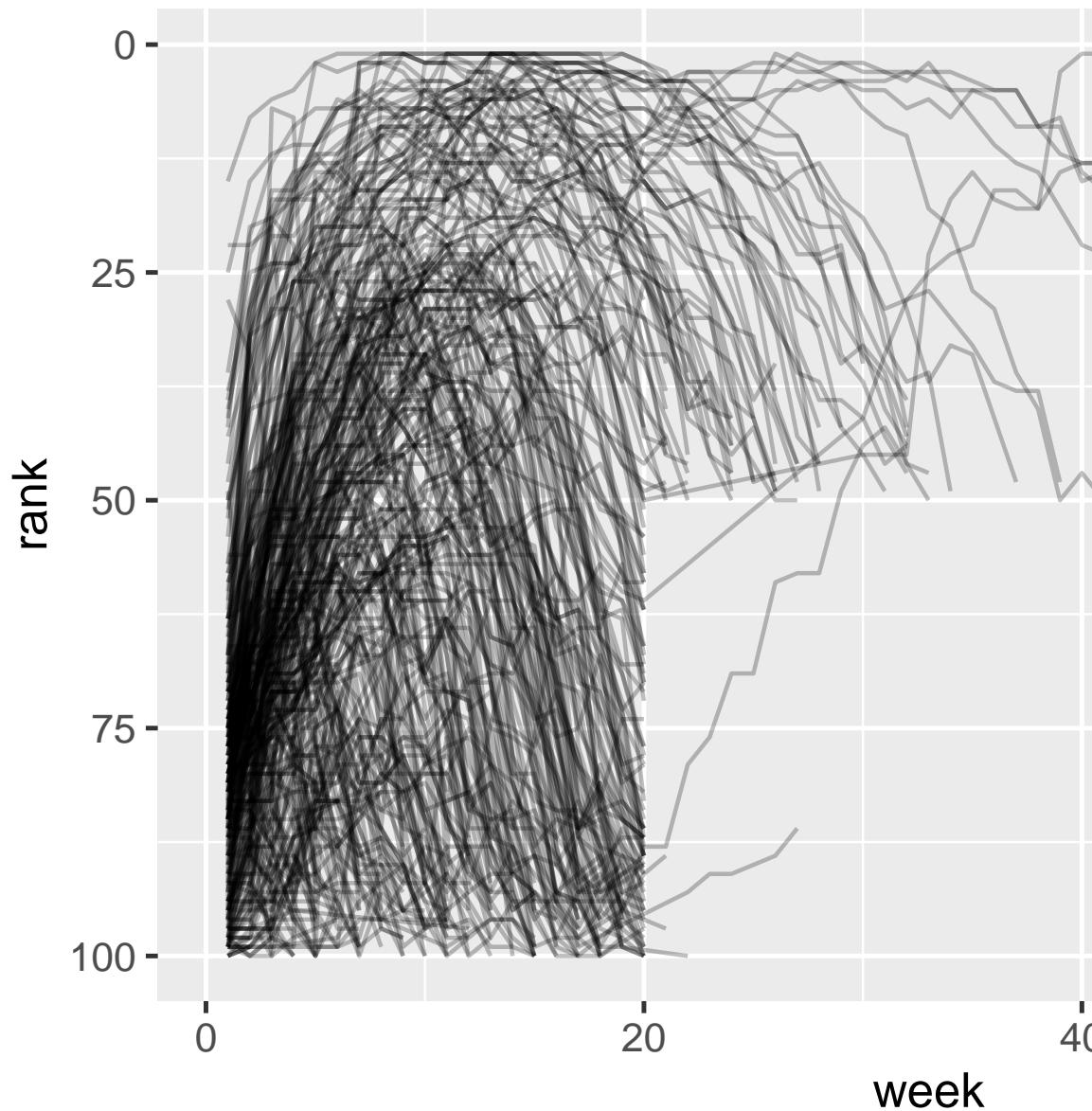


图 5.2: A line plot showing how the rank of a song changes over time.

### 5.3.2 pivoting 是如何工作的？

既然你已经看到了我们如何使用 pivoting 来重塑我们的数据，让我们花点时间来直观地理解 pivoting 对数据做了什么。让我们从一个非常简单的数据集开始，以便更容易地看到发生了什么。假设我们有三个患者，`id` 分别为 A、B 和 C，对每个患者进行了两次血压测量。我们使用 `tribble()` 来创建这些数据，`tribble()` 是一个易用的函数，这里通过手动构造小型 tibble 对象：

```
df <- tribble(
  ~id, ~bp1, ~bp2,
  "A", 100, 120,
  "B", 140, 115,
  "C", 120, 125
)
```

我们希望新数据集有三个变量：`id`（已存在）、`measurement`（列名）和 `value`（单元格值）。为了实现这一点，我们需要将 `df` 重塑（pivot）为更长的格式：

```
df |>
  pivot_longer(
    cols = bp1:bp2,
    names_to = "measurement",
    values_to = "value"
  )
#> # A tibble: 6 x 3
#>   id     measurement value
#>   <chr> <chr>       <dbl>
#> 1 A      bp1        100
```

## 5 数据整理

```
#> 2 A      bp2      120
#> 3 B      bp1      140
#> 4 B      bp2      115
#> 5 C      bp1      120
#> 6 C      bp2      125
```

重塑是如何工作的？如果我们按列思考，就会更容易理解。如图 ?? 所示，在原始数据集中已经是变量的列（如 id）的值需要被重复，每个被重塑的列重复一次。

The diagram illustrates the pivot operation. On the left, a wide table has three columns: 'id' (A, B, C), 'bp1' (100, 140, 120), and 'bp2' (120, 115, 125). An arrow points to the right, leading to a long table where each row corresponds to a value in the 'id' column of the original table. The long table has three columns: 'id' (A, A, B, B, C, C), 'measurement' (bp1, bp2, bp1, bp2, bp1, bp2), and 'value' (100, 120, 140, 115, 120, 125).

<b>id</b>	<b>measurement</b>	<b>value</b>
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

图 5.3: Columns that are already variables need to be repeated, once for each column that is pivoted.

列名变成新变量的值，新变量的名称由 `names_to` 定义，如 @fig-pivot-names 所示。它们需要在原始数据集的每一行中重复一次。

### 5.3 数据转换—长格式

The diagram illustrates the process of pivoting data from a wide format to a long format. On the left, a wide dataset is shown with columns for 'id' (A, B, C) and two measurements ('bp1', 'bp2'). An arrow points to the right, indicating the transformation, leading to a long dataset where each row corresponds to a measurement for a specific id. The 'measurement' column is now a value in the 'value' column.

<b>id</b>	<b>bp1</b>	<b>bp2</b>
A	100	120
B	140	115
C	120	125

→

<b>id</b>	<b>measurement</b>	<b>value</b>
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

图 5.4: The column names of pivoted columns become values in a new column.

The values need to be repeated once for each row of the original dataset.

## 5 数据整理

单元格值也变成新变量的值，新变量的名称由 `values_to` 定义。它们被逐行地展开。@ fig-pivot-values 展示了这一过程。

The diagram illustrates a pivot operation. On the left, there is a wide table with three columns: 'id' (containing 'A', 'B', 'C'), 'bp1' (containing '100', '140', '120'), and 'bp2' (containing '120', '115', '125'). A large arrow points from this table to the right, indicating the transformation process. On the right, there is a long table with three columns: 'id' (containing 'A', 'A', 'B', 'B', 'C', 'C'), 'measurement' (containing 'bp1', 'bp2', 'bp1', 'bp2', 'bp1', 'bp2'), and 'value' (containing '100', '120', '140', '115', '120', '125'). This transformation preserves the number of values while unwinding the rows by row.

<b>id</b>	<b>measurement</b>	<b>value</b>
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

图 5.5: The number of values is preserved (not repeated), but unwound row-by-row.

### 5.3.3 列名中包含多个变量

当列名中融合了多个信息片段而你希望将这些信息分散到独立的新变量中时，情况就会变得更加复杂。这里以你之前看到的 `table1` 等表的源头数据集 `who2` 为例：

```
who2
#> # A tibble: 7,240 x 58
#>   country      year sp_m_014 sp_m_1524 sp_m_2534 sp_m_3544 sp_m_4554
#>   <chr>       <dbl>    <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
```

### 5.3 数据转换—长格式

```
#> 1 Afghanistan 1980      NA      NA      NA      NA      NA
#> 2 Afghanistan 1981      NA      NA      NA      NA      NA
#> 3 Afghanistan 1982      NA      NA      NA      NA      NA
#> 4 Afghanistan 1983      NA      NA      NA      NA      NA
#> 5 Afghanistan 1984      NA      NA      NA      NA      NA
#> 6 Afghanistan 1985      NA      NA      NA      NA      NA
#> # i 7,234 more rows
#> # i 51 more variables: sp_m_5564 <dbl>, sp_m_65 <dbl>, sp_f_014 <dbl>, ...
```

这个数据集由世界卫生组织收集，记录了关于结核病诊断的信息，其中有两个列已经是变量且易于解读：`country` 和 `year`。接着是 56 个像 `sp_m_014`、`ep_m_4554` 和 `rel_m_3544` 这样的列。如果你长时间盯着这些列看就会发现一个规律。每个列名都由三部分组成，由 `_` 分隔。第一部分 `sp/rel/ep` 描述了用于诊断的方法；第二部分 `m/f` 是性别（在这个数据集中被编码为二进制变量）；第三部分，`014/1524/2534/3544/4554/5564/65` 是年龄范围（例如，`014` 代表 0-14 岁）。

因此，在这种情况下，`who2` 数据集中记录了六条信息：`country` 和 `year`（已经是列）；诊断方法、性别和年龄范围类别（包含在其他列名中），以及该类别中的患者数量（单元格值）。为了将这六条信息组织到六个单独的列中，我们使用 `pivot_longer()` 函数，并为 `names_to` 提供一个列名字符串向量，以及为 `names_sep` 提供一个指令来将原始变量名分割成片段，并为 `values_to` 提供一个列名：

```
who2 |>
  pivot_longer(
    cols = !(country:year),
    names_to = c("diagnosis", "gender", "age"),
```

## 5 数据整理

```
names_sep = "_",
values_to = "count"
)
#> # A tibble: 405,440 x 6
#>   country     year diagnosis gender age   count
#>   <chr>      <dbl> <chr>    <chr>  <chr> <dbl>
#> 1 Afghanistan 1980 sp        m      014     NA
#> 2 Afghanistan 1980 sp        m      1524    NA
#> 3 Afghanistan 1980 sp        m      2534    NA
#> 4 Afghanistan 1980 sp        m      3544    NA
#> 5 Afghanistan 1980 sp        m      4554    NA
#> 6 Afghanistan 1980 sp        m      5564    NA
#> # i 405,434 more rows
```

除了 `names_sep` 之外，你还可以使用 `names_pattern`。当你在 @sec-regular-expressions 学习了正则表达式之后，就可以使用它来从更复杂的命名场景中提取变量。

从概念上讲，这只是在你之前看到的简单情况上做了微小的变化。@ sec-regular-expressions 展示了基本思想：现在，列名不再只是转换为一个单独的列，而是转换为多个列。你可以想象这个过程分为两步（首先转换然后分割），但实际上它是在一步中完成的，因为这样更快。

### 5.3.4 列标题中的数据和变量名

接下来更复杂的步骤是当列名包含变量值和变量名的混合时。例如，以 `household` 数据集为例：

### 5.3 数据转换—长格式

The diagram illustrates the process of pivoting columns. On the left, a wide-format table with columns `id`, `x_1`, and `y_2` is shown. The rows contain data for three entities: A, B, and C. The values in the `x_1` and `y_2` columns are highlighted in green and blue respectively. An arrow points from this table to a long-format table on the right. The long-format table has four columns: `id`, `name`, `number`, and `value`. It contains six rows, where each entity (A, B, C) is represented by two rows. The first row for each entity has `name` as 'x' and `number` as 1, with the value in `value` being the corresponding value from the `x_1` column of the wide table. The second row for each entity has `name` as 'y' and `number` as 2, with the value in `value` being the corresponding value from the `y_2` column of the wide table.

<code>id</code>	<code>x_1</code>	<code>y_2</code>	
A	1	2	
B	3	4	
C	5	6	

→

<code>id</code>	<code>name</code>	<code>number</code>	<code>value</code>
A	x	1	1
A	y	2	2
B	x	1	3
B	y	2	4
C	x	1	5
C	y	2	6

图 5.6: Pivoting columns with multiple pieces of information in the names means that each column name now fills in values in multiple output columns.

## 5 数据整理

```
household
#> # A tibble: 5 x 5
#>   family dob_child1 dob_child2 name_child1 name_child2
#>   <int> <date>     <date>     <chr>      <chr>
#> 1     1 1998-11-26 2000-01-29 Susan       Jose
#> 2     2 1996-06-22 NA          Mark        <NA>
#> 3     3 2002-07-11 2004-04-05 Sam         Seth
#> 4     4 2004-10-10 2009-08-27 Craig      Khai
#> 5     5 2000-12-05 2005-02-28 Parker    Gracie
```

这个数据集包含了五个家庭的数据，每个家庭最多有两个孩子的姓名和出生日期。数据集中的新挑战是列名包含了两个变量的名称（`dob`、`name`）和另一个变量（`child`, 其值为 1 或 2）的值。为了解决这个问题，我们再次需要向 `names_to` 提供一个向量，但这次我们使用特殊的".value" 哨兵值（sentinel）；这不是变量的名称，而是一个独特的值，它告诉 `pivot_longer()` 做一些不同的事情。这会覆盖通常的 `values_to` 参数，使用重塑后的列名的第一个组成部分作为输出中的变量名。

```
household |>
  pivot_longer(
    cols = !family,
    names_to = c(".value", "child"),
    names_sep = "_",
    values_drop_na = TRUE
  )
#> # A tibble: 9 x 4
#>   family child   dob       name
#>   <int> <chr> <date>     <chr>
```

### 5.3 数据转换—长格式

```
#> 1      1 child1 1998-11-26 Susan
#> 2      1 child2 2000-01-29 Jose
#> 3      2 child1 1996-06-22 Mark
#> 4      3 child1 2002-07-11 Sam
#> 5      3 child2 2004-04-05 Seth
#> 6      4 child1 2004-10-10 Craig
#> # i 3 more rows
```

由于输入数据的格式，必须创建一些明确的缺失变量（例如，对于只有一个孩子的家庭）通过设置 `values_drop_na = TRUE`，我们可以选择忽略这些由于数据不完整而产生的 NA 值。。

图 ?? 通过一个更简单的示例说明了基本思想。当您在 `names_to` 中使用`".value"` 时，输入中的列名对输出中的值和变量名都有贡献。

The diagram illustrates the pivoting process. On the left, a wide-format table has columns `id`, `x_1`, `x_2`, `y_1`, and `y_2`. An arrow points to the right, leading to a long-format table with columns `id`, `x`, `y`, and `num`. The first row of the wide table becomes the first row of the long table, where `x` is 1, `y` is 3, and `num` is 1. The second row of the wide table becomes the second row of the long table, where `x` is 2, `y` is 4, and `num` is 2. The third row of the wide table becomes the fourth row of the long table, where `x` is 5, `y` is 7, and `num` is 1. The fourth row of the wide table becomes the fifth row of the long table, where `x` is 6, `y` is 8, and `num` is 2.

<code>id</code>	<code>x_1</code>	<code>x_2</code>	<code>y_1</code>	<code>y_2</code>
A	1	2	3	4
B	5	6	7	8

→

<code>id</code>	<code>x</code>	<code>y</code>	<code>num</code>
A	1	3	1
A	2	4	2
B	5	7	1
B	6	8	2

图 5.7: Pivoting with `names_to = c(".value", "num")` splits the column names into two components: the first part determines the output column name (`x` or `y`)，and the second part determines the value of the `num` column.

## 5.4 数据转换——宽格式

到目前为止，我们已经使用 `pivot_longer()` 函数来解决了一个常见问题，即值最终出现在列名中的情况。接下来，我们将转置（HA HA，这里是一个双关语，因为 pivot 在英语中也有“转置”的意思）到 `pivot_wider()` 函数，该函数通过增加列和减少行来使数据集变宽，这在一个观测分布在多行时特别有用。这种情况在实际数据中较少出现，但在处理政府数据时似乎很常见。

我们将首先查看 `cms_patient_experience` 数据集，这是一个来自医疗保险和医疗补助服务中心（Centers of Medicare and Medicaid Services）的数据集，该数据集收集了关于患者体验的数据：

```
cms_patient_experience
#> # A tibble: 500 x 5
#>   org_pac_id org_nm          measure_cd  measure_title  prf_rate
#>   <chr>      <chr>          <chr>        <chr>           <dbl>
#> 1 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_1 CAHPS for MIPS~ 63
#> 2 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_2 CAHPS for MIPS~ 87
#> 3 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_3 CAHPS for MIPS~ 80
#> 4 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_5 CAHPS for MIPS~ 57
#> 5 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_8 CAHPS for MIPS~ 85
#> 6 0446157747 USC CARE MEDICAL GROUP INC CAHPS_GRP_12 CAHPS for MIPS~ 24
#> # i 494 more rows
```

被研究的核心单位是组织，但每个组织都分布在六行中，每行代表调查组织中的一个测量值。我们可以通过使用 `distinct()` 函数来查看 `measure_cd` 和 `measure_title` 值的完整集。

## 5.4 数据转换—宽格式

```
cms_patient_experience |>
  distinct(measure_cd, measure_title)
#> # A tibble: 6 x 2
#>   measure_cd    measure_title
#>   <chr>         <chr>
#> 1 CAHPS_GRP_1  CAHPS for MIPS SSM: Getting Timely Care, Appointments, and In-
#> 2 CAHPS_GRP_2  CAHPS for MIPS SSM: How Well Providers Communicate
#> 3 CAHPS_GRP_3  CAHPS for MIPS SSM: Patient's Rating of Provider
#> 4 CAHPS_GRP_5  CAHPS for MIPS SSM: Health Promotion and Education
#> 5 CAHPS_GRP_8  CAHPS for MIPS SSM: Courteous and Helpful Office Staff
#> 6 CAHPS_GRP_12 CAHPS for MIPS SSM: Stewardship of Patient Resources
```

这两列都不会成为特别好的变量名：`measure_cd` 没有暗示变量的含义，而 `measure_title` 是一个包含空格的长句子。目前我们将使用 `measure_cd` 作为新列名的来源，但在实际分析中，你可能希望创建既简短又有意义的变量名。

`pivot_wider()` 与 `pivot_longer()` 的操作正好相反。它不需要定义新的列名，而是选择一个现有列来提供值（`values_from`），并选择另一个列来定义新的列名（`names_from`）。

```
cms_patient_experience |>
  pivot_wider(
    names_from = measure_cd,
    values_from = prf_rate
  )
#> # A tibble: 500 x 9
#>   org_pac_id org_nm          measure_title  CAHPS_GRP_1  CAHPS_GRP_2
#>   <chr>      <chr>         <chr>           <dbl>        <dbl>
```

## 5 数据整理

```
#> 1 0446157747 USC CARE MEDICAL GROUP ~ CAHPS for MIPS~ 63
#> 2 0446157747 USC CARE MEDICAL GROUP ~ CAHPS for MIPS~ NA
#> 3 0446157747 USC CARE MEDICAL GROUP ~ CAHPS for MIPS~ NA
#> 4 0446157747 USC CARE MEDICAL GROUP ~ CAHPS for MIPS~ NA
#> 5 0446157747 USC CARE MEDICAL GROUP ~ CAHPS for MIPS~ NA
#> 6 0446157747 USC CARE MEDICAL GROUP ~ CAHPS for MIPS~ NA
#> # i 494 more rows
#> # i 4 more variables: CAHPS_GRP_3 <dbl>, CAHPS_GRP_5 <dbl>, ...
```

输出的结果看起来不太对，因为似乎每个组织仍然有多行数据。这是因为我们还需告诉 `pivot_wider()` 哪个列或哪些列的值能够唯一地标识每一行；在这个例子中，这些是以 "org" 开头的变量。

```
cms_patient_experience |>
  pivot_wider(
    id_cols = starts_with("org"),
    names_from = measure_cd,
    values_from = prf_rate
  )
#> # A tibble: 95 x 8
#>   org_pac_id org_nm      CAHPS_GRP_1 CAHPS_GRP_2 CAHPS_GRP_3 CAHPS_GRP_5
#>   <chr>     <chr>       <dbl>       <dbl>       <dbl>       <dbl>
#> 1 0446157747 USC CARE MEDICA~       63         87         86         57
#> 2 0446162697 ASSOCIATION OF ~       59         85         83         63
#> 3 0547164295 BEAVER MEDICAL ~       49         NA         75         49
#> 4 0749333730 CAPE PHYSICIANS~       67         84         85         65
#> 5 0840104360 ALLIANCE PHYSIC~       66         87         87         64
#> 6 0840109864 REX HOSPITAL INC       73         87         84         64
```

## 5.4 数据转换—宽格式

```
#> # i 89 more rows
#> # i 2 more variables: CAHPS_GRP_8 <dbl>, CAHPS_GRP_12 <dbl>
```

这就给出了我们想要的输出。

### 5.4.1 pivot\_wider() 是如何工作的?

为了理解 `pivot_wider()` 是如何工作的，让我们再次从一个非常简单的数据集开始。这次我们有两个患者，`id` 分别为 A 和 B，我们在患者 A 上进行了三次血压测量，在患者 B 上进行了两次测量：

```
df <- tribble(
  ~id, ~measurement, ~value,
  "A",      "bp1",    100,
  "B",      "bp1",    140,
  "B",      "bp2",    115,
  "A",      "bp2",    120,
  "A",      "bp3",    105
)
```

我们将从 `value` 列中获取值，从 `measurement` 列中获取名称：

```
df |>
  pivot_wider(
    names_from = measurement,
    values_from = value
  )
```

## 5 数据整理

```
#> # A tibble: 2 x 4
#>   id      bp1    bp2    bp3
#>   <chr> <dbl> <dbl> <dbl>
#> 1 A        100    120    105
#> 2 B        140    115     NA
```

要开始这个过程，`pivot_wider()` 首先需要弄清楚行和列中的内容。新的列名将是 `measurement` 的唯一值。

```
df |>
  distinct(measurement) |>
  pull()
#> [1] "bp1" "bp2" "bp3"
```

默认情况下，输出中的行由不包含在新名称或值中的所有变量决定。这些被称为 `id_cols`。这里只有一列，但通常可以是任意数量的列。

```
df |>
  select(-measurement, -value) |>
  distinct()
#> # A tibble: 2 x 1
#>   id
#>   <chr>
#> 1 A
#> 2 B
```

然后 `pivot_wider()` 将这些结果组合起来生成一个空数据框：

```
df |>
  select(-measurement, -value) |>
  distinct() |>
  mutate(x = NA, y = NA, z = NA)
#> # A tibble: 2 x 4
#>   id     x     y     z
#>   <chr> <lgl> <lgl> <lgl>
#> 1 A     NA    NA    NA
#> 2 B     NA    NA    NA
```

然后，它使用输入中的数据填充所有缺失的值。在这个例子中，输出中的每个单元格在输入中并非都有对应的值，因为患者 B 没有第三次血压测量，所以那个单元格的值是缺失的。我们将在章节 ?? 中探讨 `pivot_wider()` 可以“制造”缺失值这个观点。

你可能还会想，如果输入中有多个行对应于输出中的一个单元格会发生什么。下面的例子中有两行对应于 `id` 为 “A” 和 `measurement` 为 “bp1”的单元格：

```
df <- tribble(
  ~id, ~measurement, ~value,
  "A",      "bp1",    100,
  "A",      "bp1",    102,
  "A",      "bp2",    120,
  "B",      "bp1",    140,
  "B",      "bp2",    115
)
```

## 5 数据整理

如果我们尝试对这样的数据集进行重塑，会得到一个包含列表-列的输出，你将在 @sec-rectangling 中学习更多关于列表-列的内容：

```
df |>
  pivot_wider(
    names_from = measurement,
    values_from = value
  )
#> Warning: Values from `value` are not uniquely identified; output will contain
#> list-cols.
#> * Use `values_fn = list` to suppress this warning.
#> * Use `values_fn = {summary_fun}` to summarise duplicates.
#> * Use the following dplyr code to identify duplicates.
#>   {data} |>
#>     dplyr::summarise(n = dplyr::n(), .by = c(id, measurement)) |>
#>     dplyr::filter(n > 1L)
#> # A tibble: 2 x 3
#>   id     bp1      bp2
#>   <chr> <list>   <list>
#> 1 A     <dbl [2]> <dbl [1]>
#> 2 B     <dbl [1]> <dbl [1]>
```

由于你还不知道如何处理这类数据，因此需要遵循警告中的提示来找出问题所在：

```
df |>
  group_by(id, measurement) |>
  summarize(n = n(), .groups = "drop") |>
```

```
filter(n > 1)
#> # A tibble: 1 x 3
#>   id    measurement     n
#>   <chr> <chr>       <int>
#> 1 A      bp1          2
```

接下来你需要找出你的数据出了什么问题，然后修复潜在的损坏，或者使用你的分组和汇总技能来确保每个行和列值的组合都只有一行。

## 5.5 小结

在本章中，你学习了整整齐数据：将数据变量放在列中，观测放在行中。整齐数据使得在 tidyverse 环境中工作更加容易，因为它是一种被大多数函数所理解的统一结构。主要的挑战是将你从任何结构中接收到的数据转换为整齐格式。为此，你学习了 `pivot_longer()` 和 `pivot_wider()` 函数，它们允许你整理许多不整齐的数据集。我们在这里给出的例子是从 `vignette("pivot", package = "tidyverse")` 中挑选出来的，因此如果你遇到本章没有帮助你解决的问题，那么 vignette 是一个进行尝试的好地方。

另一个挑战是，对于给定的数据集，可能无法将更长或更宽的版本标记为“整齐”的，这在一定程度上反映了我们对整洁数据的定义。我们说整齐数据在每列中都有一个变量，但我们实际上并没有定义什么是变量（而且实际上很难这样做）。务实地讲，变量就是使你的分析最容易进行的任何东西。因此，如果你正在为如何进行某些计算而苦恼，请考虑更改你的数据组织方式；不要害怕在需要时取消整齐化、转换和重新整齐化！

## 5 数据整理

如果你喜欢这一章并想了解更多底层理论，你可以学习发表在《Journal of Statistical Software》上的[Tidy Data](#) 论文中的历史和理论基础。

现在你已经编写了大量的 R 代码，是时候学习如何将你的代码组织到文件和目录中了。在下一章中，你将了解脚本和项目的所有优势，以及它们提供的一些工具，这些工具将使你的生活更加轻松。

# 6 工作流程：脚本和项目

本章将向你介绍组织代码的两个基本工具：脚本和项目。

## 6.1 脚本

到目前为止，你已经使用控制台来运行代码。这是一个很好的起点，但是当你创建更复杂的 ggplot2 图形和更长的 dplyr 管道时，您会发现控制台空间很快就变得不够用了。为了给自己更多的工作空间，请使用脚本编辑器。通过点击“文件”菜单，选择“新建文件”，然后选择“R 脚本”，或者使用键盘快捷键 Cmd/Ctrl + Shift + N 来打开它。你会看到四个窗格，如图 ?? 所示。脚本编辑器是测试代码的好地方。当你想要更改某些内容时，不再需要重新输入整个代码，只需编辑脚本并重新运行它。而且，一旦编写好了符合你需求的代码，可以将其保存为脚本文件，以便以后轻松返回。

### 6.1.1 运行代码

脚本编辑器是构建复杂的 ggplot2 图形或长序列的 dplyr 操作的好地方。有效使用脚本编辑器的关键是记住一个最重要的键盘快捷键：Cmd/Ctrl + Enter。这个快捷键可以在控制台中执行当前的 R 表达式。例如，请看下面的代码。

## 6 工作流程: 脚本和项目

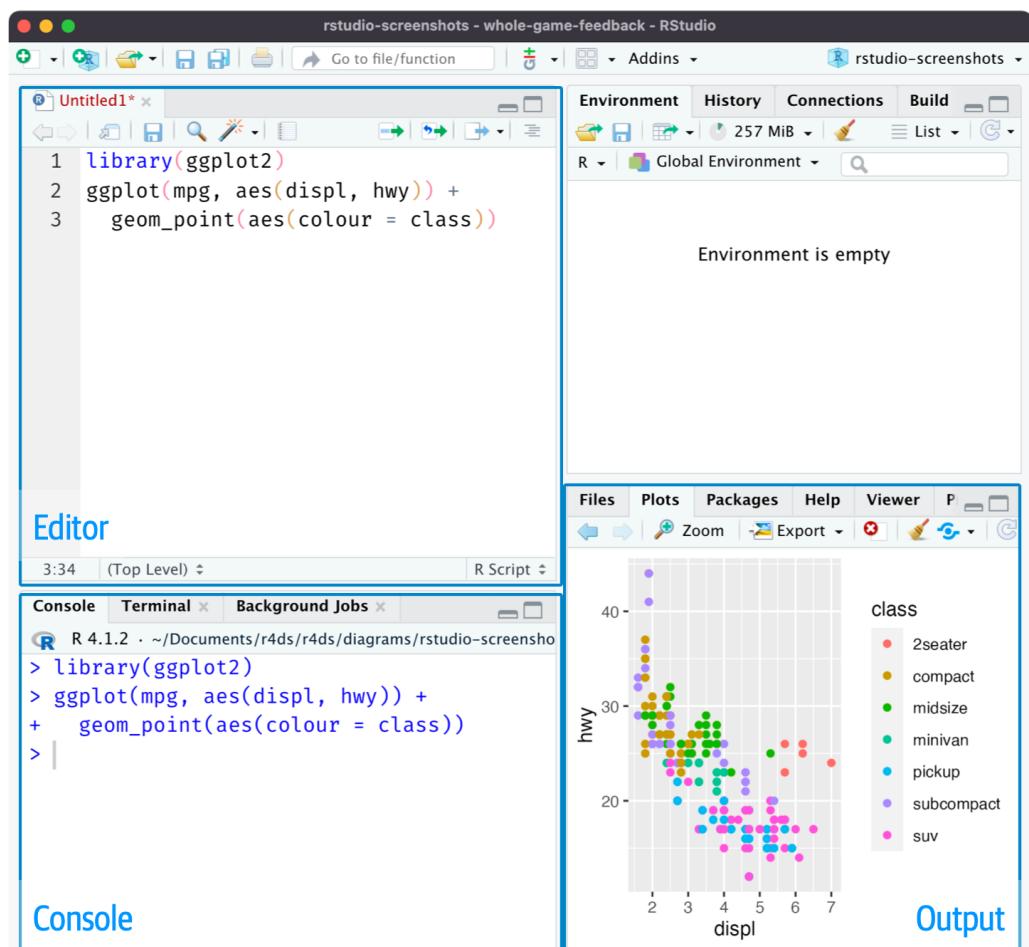


图 6.1: Opening the script editor adds a new pane at the top-left of the IDE.

## 6.1 脚本

```
library(dplyr)
library(nycflights13)

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

not_cancelled |>
  group_by(year, month, day) |>
  summarize(mean = mean(dep_delay))
```

如果你的光标位于 `|>` 处，按下 Cmd/Ctrl + Enter 将运行生成 `not_cancelled` 的完整命令。同时，光标也会移动到下一个语句（以 `not_cancelled |>` 开头）。这使得通过反复按下 Cmd/Ctrl + Enter 来逐步执行整个脚本变得很容易。

与其逐个表达式地运行代码，还可以使用 Cmd/Ctrl + Shift + S 一次性执行完整的脚本。定期这样做是保证你捕获了脚本中代码所有重要部分的好方法。

我们建议你始终在脚本开头列出所需的包。这样，如果与他人共享代码，他们可以轻松地看到他们需要安装哪些包。但是，请注意，不要在共享的脚本中包含 `install.packages()`。如果不小心将一个会更改计算机的脚本交给他们是欠考虑的！

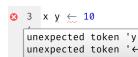
在学习后续章节时，我们强烈推荐从脚本编辑器开始并练习您的键盘快捷键。随着时间的推移，以这种方式将代码发送到控制台会变得如此自然，甚至不用去考虑。

### 6.1.2 RStudio 诊断

在脚本编辑器中, RStudio 会在侧边栏中用一条红色弯弯曲曲的线和一个叉来突出显示语法错误。



将鼠标悬停在十字上, 看看问题是什么:



RStudio 还会让您了解潜在的问题:



### 6.1.3 文件保存和命名

RStudio 在你退出时会自动保存脚本编辑器的内容, 并在重新打开时自动加载它。然而, 最好不要使用“Untitled1”, “Untitled2”, “Untitled3”等命名方式, 而是应该保存您的脚本并给它们起一个有意义的名字。

可能你会想用 `code.R` 或 `myscript.R` 来命名文件, 但在选择文件名之前应该多考虑一下。文件命名的三个重要原则是:

1. 文件名应该是机器可读的: 避免空格、符号和特殊字符, 不要依赖大小写来区分文件。
2. 文件名应该是人类可读的: 使用文件名来描述文件中的内容。
3. 文件名应该与默认排序方式兼容: 以数字开头命名文件, 以便按字母顺序排序时, 它们能按照使用的顺序排列。

## 6.1 脚本

例如，假设您在一个项目文件夹中有以下文件：

```
alternative model.R  
code for exploratory analysis.r  
finalreport.qmd  
FinalReport.qmd  
fig 1.png  
Figure_02.png  
model_first_try.R  
run-first.r  
temp.txt
```

这里存在几个问题：很难确定先运行哪个文件；文件名包含空格；有两个名称相同但大小写不同的文件（`finalreport` 与 `FinalReport`<sup>1</sup>），并且一些名称没有描述其内容（`run-first` 和 `temp`）。

下面是一种更好的命名和组织同一组文件的方法：

```
01-load-data.R  
02-exploratory-analysis.R  
03-model-approach-1.R  
04-model-approach-2.R  
fig-01.png  
fig-02.png  
report-2022-03-20.qmd  
report-2022-04-02.qmd  
report-draft-notes.txt
```

---

<sup>1</sup>如果你在名字中使用“final”更是在冒险；漫画《Piled Higher and Deeper》中有一个关于此话题的有趣连载。

对关键脚本进行编号可以明确它们的运行顺序, 而一致的命名方案可以更容易地看出发生了什么变化。此外, 图表的标签也类似; 报告通过文件名中包含的日期进行区分, 而 `temp` 被重命名为 `report-draft-notes` 以更好地描述其内容。如果您在目录中有大量文件, 建议进一步组织文件, 将不同类型的文件(脚本、图表等)放在不同的目录中。

## 6.2 项目

有时, 你可能需要退出 R 做其他的事情, 稍后再回到分析任务中; 有时, 你会同时处理多个任务, 并且希望将它们分开。有时, 你需要将外部世界的数据带入 R, 并将 R 中的数值结果和图表发送回外部世界。

为了处理这些实际情况, 你需要做出两个决定:

1. 什么是真相的来源? 你会保存什么作为发生的事情的永久记录?
2. 你的分析在哪里进行?

### 6.2.1 什么是真相的来源?

作为初学者, 依靠当前环境 (Environment) 包含分析过程中创建的所有对象是无可厚非的。但是, 为了更容易地进行大型项目或与他人合作, R 脚本应该成为事实真相的来源, 你可以通过 R 脚本 (和数据文件) 重新创建环境。如果只有自己的环境, 则很难重新创建 R 脚本: 你要么必须从记忆中重新键入大量代码 (在此过程中不可避免地会犯错误), 要么必须仔细挖掘您的 R 历史记录。

为了帮助保持 R 脚本作为你分析的真相来源, 我们强烈建议你不要让 RStudio 在会话之间保留您的工作空间 (workspace)。您可以通过运行

`usethis::use_blank_slate()`<sup>2</sup>或模仿图 ?? 中所示的选项来实现。这将带给您带来一些短期的痛苦，因为现在当您重新启动 RStudio 时，它将不再记住您上次运行的代码，您创建的对象或读取的数据集也将无法使用。但是这种短期的痛苦可以避免长期的痛苦，因为它迫使你在代码中捕获所有重要的过程。没有什么比在事实发生三个月后发现你只在环境中存储了重要计算的结果，而没有在代码中存储计算更糟糕的了。

有一组非常实用的键盘快捷键组合，它们将协同工作以确保你已经在编辑器中捕获了代码的重要部分：

1. 按 Cmd/Ctrl + Shift + 0/F10 来重启 R。
2. 按 Cmd/Ctrl + Shift + S 来重新运行当前脚本。

我们每周都会使用这种模式数百次。

或者，如果不使用快捷键，你可以转到“会话”>“重启 R”，然后选中并重新运行当前脚本。

#### RStudio server

如果您使用的是 RStudio server，默认情况下 R 会话不会自动重启。当你关闭 RStudio server 标签页时，可能会感觉正在关闭 R，但实际上服务器会在后台保持其运行状态。下次你返回时，你将处于和离开时完全相同的位置。这使得定期重启 R 变得尤为重要，以便你是从一个干净状态开始的。

<sup>2</sup>如果没有安装 usethis，可以使用 `install.packages("usethis")` 安装。

## 6 工作流程: 脚本和项目

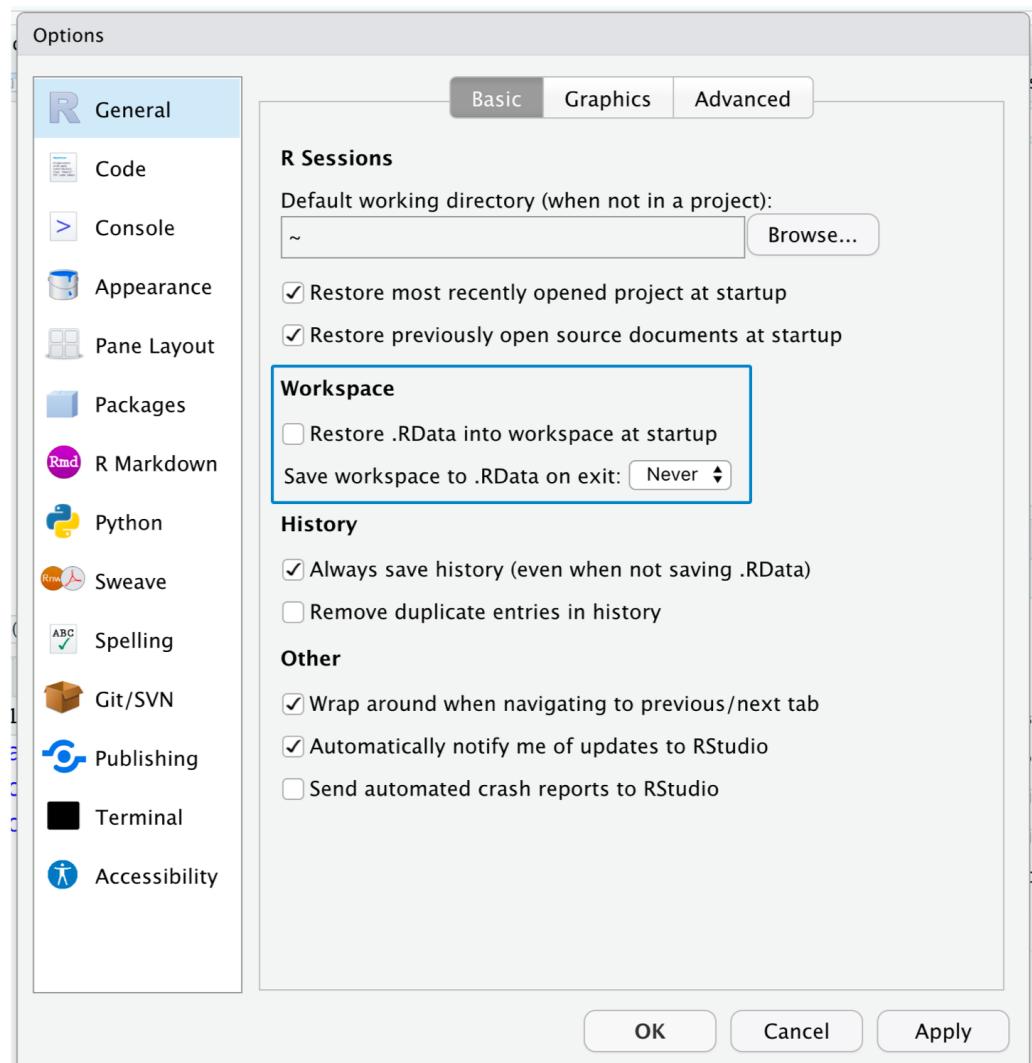
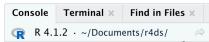


图 6.2: Copy these options in your RStudio options to always start your RStudio session with a clean slate.

### 6.2.2 分析在哪里进行？

工作目录是 R 中非常重要的概念。R 在这个目录中查找你想要加载的文件，也将要保存的文件存放在那里。R Studio 在控制台顶部显示当前的工作目录：



可以通过运行 `getwd()` 函数在 R 代码中输出当前的工作目录：

```
getwd()
#> [1] "/Users/hadley/Documents/r4ds"
```

本次 R 会话中，当前工作目录（可以将其视为“主目录”）位于 Hadley 的 Documents 文件夹中一个名为 r4ds 的子文件夹中。当您运行此代码时，它将返回不同的结果，因为你的计算机的目录结构与 Hadley 的不同！

作为 R 的初学者，让工作目录成为你的主目录、文档目录或者计算机上的任何其他奇怪的目录都是可以的。但是，你已经阅读了本书中的多个章节，不再是初学者了。现在，你应该开始将项目组织到目录中，并在处理项目时将 R 的工作目录设置为相关目录。

你可以在 R 内部设置工作目录，但不推荐这样做：

```
setwd("/path/to/my/CoolProject")
```

有一种更好的方法，这种方法也能让您像专家一样管理 R 工作。那就是使用 RStudio 项目。

### 6.2.3 RStudio 项目

将所有与某个项目相关的文件（输入数据、R 脚本、分析结果和图表）保存在一个目录中是一种明智且常见的做法，RStudio 通过项目提供了内置支持。让我们为您创建一个项目，以便您在使用本书的其余部分时可以使用它。点击“文件”>“新建项目”，然后按照 @fig-new-project 中显示的步骤操作。

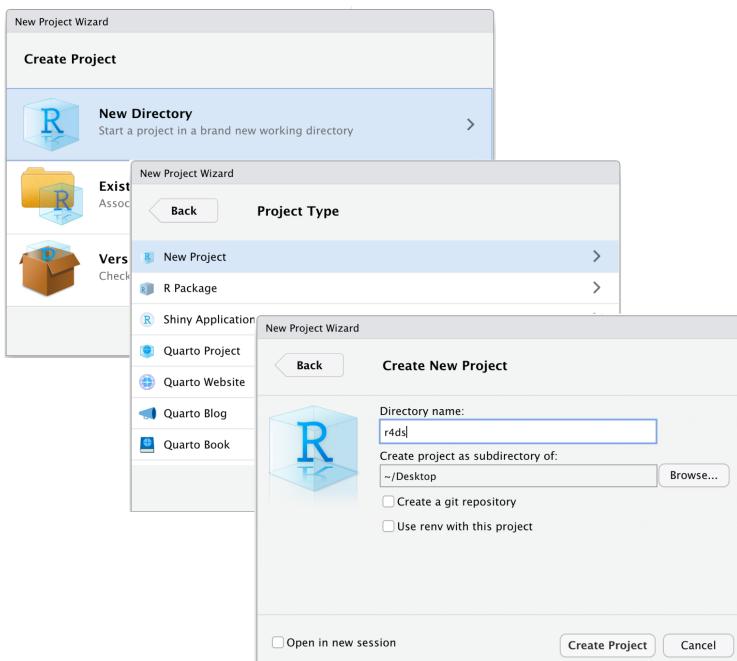


图 6.3: To create new project: (top) first click New Directory, then (middle) click New Project, then (bottom) fill in the directory (project) name, choose a good subdirectory for its home and click Create Project.

将你的项目命名为 `r4ds`，并仔细考虑将其放在哪个子目录中。如果不将其存储在一个合理的地方，将来会很难找到它！

一旦完成此过程，你将获得一个专为此书创建的新 RStudio 项目。请检查项目的“主目录”是否已设置为当前工作目录：

```
getwd()  
#> [1] /Users/hadley/Documents/r4ds
```

现在，在脚本编辑器中输入以下命令并将文件保存为“diamonds.R”。然后，创建一个名为“data”的新文件夹。你可以通过在 RStudio 的“文件”面板中点击“新建文件夹”按钮来完成此操作。最后，运行完整的脚本，它将把 PNG 和 CSV 文件保存到您的项目目录中。不用关注细节，你将在本书的后续中学习它们。

```
library(tidyverse)  
  
ggplot(diamonds, aes(x = carat, y = price)) +  
  geom_hex()  
ggsave("diamonds.png")  
  
write_csv(diamonds, "data/diamonds.csv")
```

退出 RStudio。检查与你的项目关联的文件夹——注意.Rproj 文件。双击该文件以重新打开项目。请注意，你回到了离开时的位置：工作目录和命令历史与原来相同，你正在处理的所有文件仍然打开。但是，由于遵循了上面的说明，你将拥有一个全新的环境，确保你是从一个干净的状态开始的。

根据操作系统的不同，选择你最喜欢的方式在计算机上搜索 diamonds.png。你会找到 PNG 文件（这并不奇怪），还会找到创建它的脚本文件 diamonds.R。这是一个巨大的胜利！有一天，你可能会想要重新生成一个图形或只是了解它

## 6 工作流程: 脚本和项目

的来源。如果你严格地使用 R 代码将图形保存到文件中而从不使用鼠标或剪贴板, 那么你将能够轻松地重复以前的工作!

### 6.2.4 相对和绝对路径

一旦进入一个项目, 你应该只使用相对路径而不是绝对路径。有什么区别呢? 相对路径是相对于工作目录的, 即项目的根目录。当 Hadley 在上面写 `data/diamonds.csv` 时, 它是 `/Users/hadley/Documents/r4ds/data/diamonds.csv` 的快捷方式。但如果 Mine 在她的计算机上运行这段代码, 它将指向 `/Users/Mine/Documents/r4ds/data/diamonds.csv`。这就是为什么相对路径很重要: 无论 R 项目文件夹最终位于何处, 它们都能正常工作。

绝对路径是指向同一个位置, 与你的工作目录无关。根据操作系统不同, 它们看上去会有所不同。在 Windows 上, 它们以驱动器字母 (例如, C:) 或两个反斜杠 (例如, \\servername) 开头, 而在 Mac/Linux 上, 它们以斜杠 “/” 开头 (例如, /users/hadley)。你不应该在脚本中使用绝对路径, 因为它们会妨碍共享: 没有人会拥有与你完全相同的目录配置。

操作系统之间还有一个重要的区别: 如何分隔路径的各个部分。Mac 和 Linux 使用斜杠 (例如, `data/diamonds.csv`), 而 Windows 使用反斜杠 (例如, `data\iamonds.csv`)。R 可以与这两种类型一起工作 (无论你当前使用什么平台), 但不幸的是, 反斜杠在 R 中有特殊含义, 要在路径中获得单个反斜杠, 您需要键入两个反斜杠! 这会使生活变得令人沮丧, 因此我们建议使用 Linux/Mac 风格的斜杠。

## 6.3 练习

1. 登录 RStudio Tips Twitter 账户 <https://twitter.com/rstudiotips>, 找到一个有趣的技巧, 练习使用它!
2. RStudio 诊断还会报告哪些常见错误? 阅读 <https://support.posit.co/hc/en-us/articles/205753617-Code-Diagnostics> 找到答案。

## 6.4 小结

在本章中, 你学习了如何在脚本 (文件) 和项目 (目录) 中组织代码。就像代码风格一样, 这一开始可能会感觉像是在做无用功。但是, 随着你在多个项目中积累更多的代码, 你会开始认识到前期的一点组织工作可以为你节省大量的时间。

总的来说, 脚本和项目为你提供了一个坚实的工作流程, 这会在未来很好地为你服务:

- 为每个数据分析项目创建一个 RStudio 项目;
- 在项目中保存你的脚本 (起一个带有信息的名字), 编辑它们, 分部分或整体运行它们; 经常重新启动 R 以确保你已将所有内容捕获到脚本中。
- 永远只使用相对路径, 而不是绝对路径。

然后, 你需要的所有内容都存放于一个地方, 并且与您正在处理的所有其他项目干净地分隔开。

到目前为止, 我们一直在使用 R 包中包含的数据集。这使得在预先准备的数据上进行一些实践变得更容易, 但显然你的数据不会以这种方式提供。因此, 在下一章中将学习如何使用 `readr` 包将数据从磁盘加载到 R 会话中。



# 7 数据导入

## 7.1 引言

使用 R 包提供的数据来学习数据科学工具是一个很好的方法，但总有一天你会想将所学应用于自己的数据。在本章中，你将学习将数据文件读入 R 的基础知识。

具体来说，本章将重点关注读取纯文本矩形文件。我们将从处理列名、类型和缺失数据的实用建议开始。然后，学习如何一次从多个文件中读取数据，以及如何将 R 中的数据写入文件。最后，学习如何在 R 中手动创建数据框。

### 7.1.1 必要条件

在本章中，你将学习如何使用 `readr` 包在 R 中加载平面文件，`readr` 包是核心包 `tidyverse` 的一部分。

```
library(tidyverse)
```

## 7.2 从文件读取数据

首先, 我们将关注最常见的矩形数据文件类型: CSV (Comma-Separated Values 的缩写)。下面是一个简单的 CSV 文件, 第一行通常被称为标题行, 给出了列名, 接下来的六行提供了数据, 列之间由逗号分隔, 也称为定界符。

```
Student ID,Full Name,favourite.food,mealPlan,AGE
1,Sunil Huffmann,Strawberry yoghurt,Lunch only,4
2,Barclay Lynn,French fries,Lunch only,5
3,Jayendra Lyne,N/A,Breakfast and lunch,7
4,Leon Rossini,Anchovies,Lunch only,
5,Chidiegwu Dunkel,Pizza,Breakfast and lunch,five
6,Güvenç Attila,Ice cream,Lunch only,6
```

表 ?? 以表格形式呈现了相同的数据。

表 7.1: Data from the students.csv file as a table.

Student				
ID	Full Name	favourite.food	mealPlan	AGE
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4
2	Barclay Lynn	French fries	Lunch only	5
3	Jayendra Lyne	N/A	Breakfast and lunch	7
4	Leon Rossini	Anchovies	Lunch only	NA
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five

## 7.2 从文件读取数据

表 7.1: Data from the students.csv file as a table.

Student	ID	Full Name	favourite.food	mealPlan	AGE
	6	Güvenç Attila	Ice cream	Lunch only	6

我们使用 `read_csv()` 将这个文件读入 R。第一个参数是最重要的：文件路径。可以将路径视为文件的地址，文件名为 `students.csv`，它位于 `data` 文件夹中。

```
students <- read_csv("data/students.csv")
#> Rows: 6 Columns: 5
#> -- Column specification -----
#> Delimiter: ","
#> chr (4): Full Name, favourite.food, mealPlan, AGE
#> dbl (1): Student ID
#>
#> 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.
```

上面的代码在你的项目中存在名为 `data` 的文件夹并且其中包含 `students.csv` 文件时有效。您可以从<https://pos.it/r4ds-students-csv>下载 `students.csv` 文件，或者可以直接从该 URL 读取它，使用如下方式：

```
students <- read_csv("https://pos.it/r4ds-students-csv")
```

当你运行 `read_csv()` 时，它会输出一条消息，告诉你数据的行数和列数、所使用的分隔符以及列规范（按列中数据类型组织的列名）。它还会输出关于如

## 7 数据导入

何检索完整列规范和如何静默此消息的一些信息。这条消息是 `readr` 包的一个组成部分，我们将在 @sec-col-types 中再次讨论它。

### 7.2.1 实用建议

一旦你读取了数据，第一步通常是以某种方式转换它，以使其在剩余的分析中更容易处理。让我们带着这个想法再次查看 `students` 数据。

```
students
#> # A tibble: 6 x 5
#>   `Student ID` `Full Name`   favourite.food    mealPlan     AGE
#>   <dbl> <chr>           <chr>          <chr>        <chr>
#> 1      1 Sunil Huffmann  Strawberry yoghurt Lunch only  4
#> 2      2 Barclay Lynn    French fries    Lunch only  5
#> 3      3 Jayendra Lyne  N/A            Breakfast and lunch 7
#> 4      4 Leon Rossini   Anchovies    Lunch only  <NA>
#> 5      5 Chidiegwu Dunkel Pizza  Breakfast and lunch five
#> 6      6 Güvenç Attila   Ice cream    Lunch only  6
```

在 `favourite.food` 列中有一堆食品项目，然后是字符串 `N/A`，它应该是一个 R 会识别为“不可用”的真正的 `NA` 值。这是我们可以通过 `na` 参数来解决的问题。默认情况下，`read_csv()` 只将此数据集中的空字符串 ("") 识别为 `NAs`，我们希望它也能识别字符串 "`N/A`"。

```
students <- read_csv("data/students.csv", na = c("N/A", ""))
students
```

## 7.2 从文件读取数据

```
#> # A tibble: 6 x 5
#>   `Student ID` `Full Name`   favourite.food    mealPlan      AGE
#>   <dbl> <chr>           <chr>          <chr>          <chr>
#> 1     1 Sunil Huffmann  Strawberry yoghurt Lunch only    4
#> 2     2 Barclay Lynn    French fries     Lunch only    5
#> 3     3 Jayendra Lyne  <NA>            Breakfast and lunch 7
#> 4     4 Leon Rossini   Anchovies      Lunch only    <NA>
#> 5     5 Chidiegwu Dunkel Pizza    Breakfast and lunch five
#> 6     6 Güvenç Attila   Ice cream      Lunch only    6
```

你可能还注意到，`Student ID` 和 `Full Name` 列使用了反引号。这是因为它们包含空格，违反了 R 中变量名的常规规则，它们是非语法(**non-syntactic**)名称。要引用这些变量，你需要使用反引号`将它们括起来。

```
students |>
  rename(
    student_id = `Student ID`,
    full_name = `Full Name`
  )
#> # A tibble: 6 x 5
#>   student_id full_name   favourite.food    mealPlan      AGE
#>   <dbl> <chr>           <chr>          <chr>          <chr>
#> 1     1 Sunil Huffmann  Strawberry yoghurt Lunch only    4
#> 2     2 Barclay Lynn    French fries     Lunch only    5
#> 3     3 Jayendra Lyne  <NA>            Breakfast and lunch 7
#> 4     4 Leon Rossini   Anchovies      Lunch only    <NA>
#> 5     5 Chidiegwu Dunkel Pizza    Breakfast and lunch five
#> 6     6 Güvenç Attila   Ice cream      Lunch only    6
```

## 7 数据导入

另一种方法是使用 `janitor::clean_names()` 函数，该函数使用一些启发式方法来一次性将它们全部转换为蛇形命名法（snake case）<sup>1</sup>。

```
students |> janitor::clean_names()
#> # A tibble: 6 x 5
#>   student_id full_name      favourite_food meal_plan    age
#>       <dbl> <chr>          <chr>        <chr>
#> 1         1 Sunil Huffmann Strawberry yoghurt Lunch only 4
#> 2         2 Barclay Lynn    French fries   Lunch only 5
#> 3         3 Jayendra Lyne  <NA>        Breakfast and lunch 7
#> 4         4 Leon Rossini   Anchovies   Lunch only <NA>
#> 5         5 Chidiegwu Dunkel Pizza   Breakfast and lunch five
#> 6         6 Güvenç Attila   Ice cream   Lunch only 6
```

读取数据后的另一个常见任务是考虑变量类型。例如，`meal_plan` 是一个分类变量，具有一组已知的可能值，在 R 中应该表示为因子：

```
students |>
  janitor::clean_names() |>
  mutate(meal_plan = factor(meal_plan))
#> # A tibble: 6 x 5
#>   student_id full_name      favourite_food meal_plan    age
#>       <dbl> <chr>          <chr>        <fct>
#> 1         1 Sunil Huffmann Strawberry yoghurt Lunch only 4
#> 2         2 Barclay Lynn    French fries   Lunch only 5
#> 3         3 Jayendra Lyne  <NA>        Breakfast and lunch 7
```

---

<sup>1</sup>`janitor`包不是 tidyverse 的一部分，但它提供了方便的数据清理功能，并且在使用 `|>` 的数据管道中工作得很好。

## 7.2 从文件读取数据

```
#> 4      4 Leon Rossini Anchovies      Lunch only      <NA>
#> 5      5 Chidiegwu Dunkel Pizza      Breakfast and lunch five
#> 6      6 Güvenç Attila  Ice cream      Lunch only      6
```

请注意，`meal_plan` 变量中的值保持不变，但变量名下方表示的变量类型已经从字符型（`<chr>`）变为了因子型（`<fct>`）。你将在章节 `??` 中了解更多关于因子的内容。

在分析这些数据之前，你可能想要修复 `age` 和 `id` 列。目前，`age` 是一个字符型变量，因为其中一个观测值被输入为文本 `five` 而不是数字 `5`。我们将在章节 `??` 详细讨论如何修复这个问题。

```
students <- students |>
  janitor::clean_names() |>
  mutate(
    meal_plan = factor(meal_plan),
    age = parse_number(if_else(age == "five", "5", age))
  )

students
#> # A tibble: 6 x 5
#>   student_id full_name     favourite_food meal_plan      age
#>       <dbl> <chr>          <chr>           <fct>        <dbl>
#> 1          1 Sunil Huffmann Strawberry yoghurt Lunch only      4
#> 2          2 Barclay Lynn   French fries      Lunch only      5
#> 3          3 Jayendra Lyne <NA>             Breakfast and lunch  7
#> 4          4 Leon Rossini Anchovies      Lunch only      NA
```

## 7 数据导入

```
#> 5      5 Chidiegwu Dunkel Pizza      Breakfast and lunch 5
#> 6      6 Güvenç Attila     Ice cream      Lunch only 6
```

这里有一个新的函数 `if_else()`，它有三个参数。第一个参数 `test` 应该是一个逻辑向量。当 `test` 为 `TRUE` 时，结果将包含第二个参数 `yes` 的值；当它为 `FALSE` 时，将包含第三个参数 `no` 的值。在这里，我们表示如果 `age` 是字符串 `"five"`，则将其变为 `"5"`，如果不是，则保持原样。你将在章节 `??` 部分了解更多关于 `if_else()` 和逻辑向量的内容。

### 7.2.2 其他参数

这里还有一些其他重要的参数我们需要提及。如果先向你们展示一个有用的技巧，这些参数的演示就会更容易理解：`read_csv()` 可以读取你创建并格式化为 CSV 文件格式的文本字符串。

```
read_csv(
  "a,b,c
  1,2,3
  4,5,6"
)
#> # A tibble: 2 x 3
#>       a     b     c
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

通常，`read_csv()` 使用数据的第一行作为列名，这是一个非常常见的约定。但是，在文件的顶部包含几行元数据的情况也不少见。你可以使用 `skip =`

## 7.2 从文件读取数据

`n` 来跳过前 `n` 行，或者使用 `comment = "#"` 来忽略所有以（例如）`#` 开头的行：

```
read_csv(  
  "The first line of metadata  
  The second line of metadata  
  x,y,z  
  1,2,3",  
  skip = 2  
)  
#> # A tibble: 1 x 3  
#>       x     y     z  
#>   <dbl> <dbl> <dbl>  
#> 1     1     2     3  
  
read_csv(  
  "# A comment I want to skip  
  x,y,z  
  1,2,3",  
  comment = "#"  
)  
#> # A tibble: 1 x 3  
#>       x     y     z  
#>   <dbl> <dbl> <dbl>  
#> 1     1     2     3
```

在其他情况下，数据可能没有列名。你可以使用 `col_names = FALSE` 来告诉 `read_csv()` 不要将第一行作为标题，而是从 `X1` 到 `Xn` 顺序地给它们标

## 7 数据导入

记列名:

In other cases, the data might not have column names. You can use `col_names = FALSE` to tell `read_csv()` not to treat the first row as headings and instead label them sequentially from `X1` to `Xn`:

```
read_csv(  
  "1,2,3  
  4,5,6",  
  col_names = FALSE  
)  
#> # A tibble: 2 x 3  
#>   X1     X2     X3  
#>   <dbl> <dbl> <dbl>  
#> 1     1     2     3  
#> 2     4     5     6
```

另外，你可以向 `col_names` 传递一个字符向量，这个字符向量将被用作列名:

```
read_csv(  
  "1,2,3  
  4,5,6",  
  col_names = c("x", "y", "z")  
)  
#> # A tibble: 2 x 3  
#>   x     y     z  
#>   <dbl> <dbl> <dbl>
```

```
#> 1     1     2     3  
#> 2     4     5     6
```

这些参数是你需要知道的，以便读取你在实践中遇到的大多数 CSV 文件。（对于其他情况，你需要仔细检查你的 `.csv` 文件，并阅读 `read_csv()` 的许多其他参数的文档。）

### 7.2.3 其他文件类型

一旦你掌握了 `read_csv()`，使用 `readr` 的其他函数就很简单了；你只需要知道应该使用哪个函数：

- `read_csv2()` 用于读取分号分隔的文件。这些文件使用 ; 而不是 , 来分隔字段，这在使用，作为小数点标记符的国家中很常见；
- `read_tsv()` 用于读取制表符分隔 (tab-delimited) 的文件；
- `read_delim()` 用于读取具有任何分隔符的文件，如果你没有指定分隔符，它会尝试自动猜测分隔符；
- `read_fwf()` 用于读取固定宽度的文件。你可以使用 `fwf_widths()` 通过宽度指定字段，或者使用 `fwf_positions()` 通过位置指定字段；
- `read_table()` 用于读取固定宽度文件的常见变体，其中列由空白字符分隔；
- `read_log()` 用于读取 Apache 风格的日志文件。

#### 7.2.4 练习

1. 如果你想要读取一个字段之间使用 “|” 分隔的文件，你会使用什么函数？
2. 除了 `file`、`skip` 和 `comment` 之外，`read_csv()` 和 `read_tsv()` 还有哪些共同的参数？
3. `read_fwf()` 的最重要参数是什么？
4. 有时 CSV 文件中的字符串包含逗号。为了防止它们引起问题，这些字符串需要用引号字符（如 " 或 '）括起来。默认情况下，`read_csv()` 假设引号字符为 "。为了将以下文本读取到数据框中，你需要为 `read_csv()` 指定哪个参数？

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

5. 识别以下内联 CSV 文件中的每个错误是什么。当你运行代码时会发生什么？

```
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\nn\"1")
read_csv("a,b\n1,2\nna,b")
read_csv("a;b\n1;3")
```

6. 通过以下方式练习在数据框中引用非语法名称：

- a. 提取名为 1 的变量；
- b. 绘制 1 与 2 的散点图；
- c. 创建一个名为 3 的新列，该列是 2 除以 1 的结果；
- d. 将列名重命名为 `one`, `two` 和 `three`.

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

## 7.3 控制列类型

CSV 文件不包含关于每个变量类型（即它是否是逻辑型、数字型、字符串型等）的任何信息，因此 `readr` 会尝试猜测类型。本节将描述猜测过程的工作原理，如何解决一些常见的导致猜测失败的问题，以及（如果需要的话）如何自己提供列类型。最后，我们将提及一些在 `readr` 遭遇严重失败且你需要更深入地了解文件结构时非常有用的通用策略。

### 7.3.1 猜测类型

`readr` 使用一种启发式方法来确定列的类型。对于每一列，它会从第一行到最后一行均匀地抽取 1,000<sup>2</sup> 行的值，并忽略缺失值。然后，它会通过以下问题进行判断：

- 它是否只包含 `F`、`T`、`FALSE` 或 `TRUE`（忽略大小写）？如果是，则它是逻辑型；
- 它是否只包含数字（如 `1`、`-4.5`、`5e6`、`Inf`）？如果是，则它是数字型；
- 它是否符合 ISO8601 标准？如果是，则它是日期或日期-时间型。（我们将在 @sec-creating-datetime 更详细地讨论日期-时间型）；
- 否则，它一定是字符串型。

---

<sup>2</sup>你可以使用 `guess_max` 参数覆盖 1000 的默认值。

## 7 数据导入

你可以通过以下简单示例看到这个行为：

```
read_csv("logical,numeric,date,string
TRUE,1,2021-01-15,abc
false,4.5,2021-02-15,def
T,Inf,2021-02-16,ghi
")
#> # A tibble: 3 x 4
#>   logical numeric date      string
#>   <lgl>     <dbl> <date>    <chr>
#> 1 TRUE        1 2021-01-15 abc
#> 2 FALSE       4.5 2021-02-15 def
#> 3 TRUE        Inf 2021-02-16 ghi
```

如果你有一个干净的数据集，这种启发式方法很有效，但在现实生活中，你会遇到各种奇怪而美丽的失败。

### 7.3.2 缺失值、列类型和问题

列检测失败最常见的方式是某列包含了意外值，结果你得到了一个字符型列而不是更具体的类型。这种情况最常见的原因之一是缺失值，这些缺失值不是使用 `readr` 所期望的 `NA` 来记录的。

以下面这个简单的只有一列的 CSV 文件为例：

### 7.3 控制列类型

```
simple_csv <- "
x
10
.
20
30"
```

如果我们在没有任何附加参数的情况下读取它，`x` 将变成一个字符列：

```
read_csv(simple_csv)
#> # A tibble: 4 x 1
#>   x
#>   <chr>
#> 1 10
#> 2 .
#> 3 20
#> 4 30
```

在这个很小的例子中，你可以很容易地看到缺失值。但是，如果你的文件中有数千行，并且只有少数几个缺失值，并且这些缺失值分散在文件中，那么会发生什么呢？一种方法是告诉 `readr` `x` 是一列数字型数据，然后查看它在哪里失败。你可以使用 `col_types` 参数来实现这一点，该参数接受一个命名列表，其中列表的名称与 CSV 文件中的列名相匹配：

```
df <- read_csv(
  simple_csv,
  col_types = list(x = col_double())
)
```

## 7 数据导入

```
#> Warning: One or more parsing issues, call `problems()` on your data frame for
#> details, e.g.:
#>   dat <- vroom(...)
#>   problems(dat)
```

现在 `read_csv()` 报告存在问题，并告诉我们可以通过 `problems()` 找到更多信息：

```
problems(df)
#> # A tibble: 1 x 5
#>       row     col expected actual file
#>     <int> <int> <chr>    <chr>  <chr>
#> 1     3     1 a double .      /private/var/folders/2m/th7w53zx2fx6gl3g1jcj4l1
```

这告诉我们，在第 3 行、第 1 列出现了问题，`readr` 期待一个双精度浮点数，但是得到了一个`.`。这表明这个数据集使用`.`来表示缺失值。因此，我们设置 `na = ".."`，自动猜测成功，得到了我们想要的数字列：

```
read_csv(simple_csv, na = ".")
#> # A tibble: 4 x 1
#>       x
#>     <dbl>
#> 1     10
#> 2     NA
#> 3     20
#> 4     30
```

### 7.3.3 列类型

readr 总共提供了九种列类型供您使用:

- `col_logical()` 和 `col_double()` 分别用于读取逻辑值和实数。由于 readr 通常会自动为你猜测这些类型，因此它们相对不常用（除了上述情况）；
- `col_integer()` 用于读取整数。在本书中，我们很少区分整数和双精度浮点数，因为它们在功能上是等价的，但明确读取整数有时是有用的，因为它们占用的内存只有双精度浮点数的一半；
- `col_character()` 用于读取字符串。当某列是数字标识符时，明确指定它可以很有用，例如，用于标识对象的长数字序列，但对这些数字应用数学运算没有意义。示例包括电话号码、社会保障号码、信用卡号码等；
- `col_factor()`, `col_date()` 和 `col_datetime()` 分别用于创建因子、日期和日期-时间；当我们在 @sec-factors 和章节 ?? 学习这些数据类型时，你会了解更多相关信息；
- `col_number()` 是一个宽容的数字解析器，它将忽略非数字部分，特别适用于货币。在 @sec-numbers 你将了解更多相关信息；
- `col_skip()` 用于跳过某列，使其不包含在结果中；这在处理大型 CSV 文件且只想使用其中的某些列时有用，可以加快数据读取速度。

此外，也可以通过将 `list()` 切换到 `cols()` 并指定 `.default` 来覆盖默认列。

```
another_csv <- "  
x,y,z  
1,2,3"  
  
read_csv(
```

## 7 数据导入

```
another_csv,
col_types = cols(.default = col_character())
)
#> # A tibble: 1 x 3
#>   x     y     z
#>   <chr> <chr> <chr>
#> 1 1     2     3
```

Another useful helper is `cols_only()` which will read in only the columns you specify:

```
read_csv(
  another_csv,
  col_types = cols_only(x = col_character())
)
#> # A tibble: 1 x 1
#>   x
#>   <chr>
#> 1 1
```

## 7.4 从多个文件读取数据

有时，你的数据分散在多个文件中，而不是只包含在一个文件中。例如，你可能有多个月份的销售数据，每个月的数据都保存在一个单独的文件中：1月的销售数据在 `01-sales.csv` 中，2月的在 `02-sales.csv` 中，3月的在 `03-sales.csv` 中。使用 `read_csv()` 函数可以一次性读取这些数据，并将它们堆叠在一个数据框中。

## 7.4 从多个文件读取数据

```
sales_files <- c("data/01-sales.csv", "data/02-sales.csv", "data/03-sales.csv")
read_csv(sales_files, id = "file")
#> # A tibble: 19 x 6
#>   file      month   year brand item     n
#>   <chr>     <chr>   <dbl> <dbl> <dbl> <dbl>
#> 1 data/01-sales.csv January 2019    1  1234    3
#> 2 data/01-sales.csv January 2019    1  8721    9
#> 3 data/01-sales.csv January 2019    1  1822    2
#> 4 data/01-sales.csv January 2019    2  3333    1
#> 5 data/01-sales.csv January 2019    2  2156    9
#> 6 data/01-sales.csv January 2019    2  3987    6
#> # i 13 more rows
```

同样，如果在项目的 `data` 文件夹中有 CSV 文件，上述代码就可以工作。你可以从<https://pos.it/r4ds-01-sales>, <https://pos.it/r4ds-02-sales>和<https://pos.it/r4ds-03-sales>下载这些文件，或者你可以直接读取它们：

```
sales_files <- c(
  "https://pos.it/r4ds-01-sales",
  "https://pos.it/r4ds-02-sales",
  "https://pos.it/r4ds-03-sales"
)
read_csv(sales_files, id = "file")
```

`id` 参数会在结果数据框中添加一个名为 `file` 的新列，用于标识数据来自哪个文件。这在读取的文件没有包含可以帮助你追踪观测回到其原始来源的标识列时特别有用。

## 7 数据导入

如果你有很多文件想要读取，一个个写出它们的名称可能会很麻烦。相反，你可以使用基础函数 `list.files()` 来通过匹配文件名中的模式找到这些文件。你将在 @sec-regular-expressions 部分学习更多关于这些模式的知识。

```
sales_files <- list.files("data", pattern = "sales\\*.csv$", full.names = TRUE)
sales_files
#> [1] "data/01-sales.csv" "data/02-sales.csv" "data/03-sales.csv"
```

## 7.5 写入文件

`readr` 还提供了两个有用的函数，用于将数据写回磁盘：`write_csv()` 和 `write_tsv()`。这些函数最重要的参数是 `x`(要保存的数据框) 和 `file`(要保存的位置)。你还可以指定如何使用 `na` 来写入缺失值以及是否希望将其附加到现有文件中。

```
write_csv(students, "students.csv")
```

现在让我们读入 csv 文件。请注意，当你保存到 CSV 时，刚刚设置的变量类型信息会丢失，因为你重新开始从纯文本文件读取：

```
students
#> # A tibble: 6 x 5
#>   student_id full_name      favourite_food    meal_plan      age
#>       <dbl> <chr>          <chr>           <fct>        <dbl>
#> 1       1 Sunil Huffmann  Strawberry yoghurt Lunch only     4
#> 2       2 Barclay Lynn    French fries      Lunch only     5
#> 3       3 Jayendra Lyne  <NA>            Breakfast and lunch 7
```

## 7.5 写入文件

```
#> 4      4 Leon Rossini Anchovies      Lunch only      NA
#> 5      5 Chidiegwu Dunkel Pizza      Breakfast and lunch  5
#> 6      6 Güvenç Attila  Ice cream      Lunch only      6
write_csv(students, "students-2.csv")
read_csv("students-2.csv")
#> # A tibble: 6 x 5
#>   student_id full_name   favourite_food meal_plan    age
#>       <dbl> <chr>        <chr>          <chr>        <dbl>
#> 1       1 Sunil Huffmann Strawberry yoghurt Lunch only     4
#> 2       2 Barclay Lynn   French fries    Lunch only     5
#> 3       3 Jayendra Lyne <NA>           Breakfast and lunch  7
#> 4       4 Leon Rossini Anchovies      Lunch only      NA
#> 5       5 Chidiegwu Dunkel Pizza      Breakfast and lunch  5
#> 6       6 Güvenç Attila  Ice cream      Lunch only      6
```

这使得 CSV 在缓存中间结果时有些不可靠，每次加载时都需要重新创建列规范。主要有两种替代方案：

1. `write_rds()` 和 `read_rds()` 是围绕基础函数 `readRDS()` 和 `saveRDS()` 的统一包装器。这些函数将数据存储在 R 的自定义二进制格式 RDS 中。这意味着当你重新加载对象时，你加载的是与存储时完全相同的 R 对象。

```
write_rds(students, "students.rds")
read_rds("students.rds")
#> # A tibble: 6 x 5
#>   student_id full_name   favourite_food meal_plan    age
#>       <dbl> <chr>        <chr>          <chr>        <dbl>
```

## 7 数据导入

```
#> 1      1 Sunil Huffmann  Strawberry yoghurt Lunch only
#> 2      2 Barclay Lynn    French fries     Lunch only
#> 3      3 Jayendra Lyne   <NA>           Breakfast and lunch
#> 4      4 Leon Rossini    Anchovies      Lunch only
#> 5      5 Chidiegwu Dunkel Pizza       Breakfast and lunch
#> 6      6 Güvenç Attila   Ice cream      Lunch only
```

2. `arrow` 包允许你读取和写入 parquet 文件，这是一种可以快速跨编程语言共享的二进制文件格式。我们将在章节 ?? 中更深入地探讨 `arrow`。

```
library(arrow)
write_parquet(students, "students.parquet")
read_parquet("students.parquet")
#> # A tibble: 6 × 5
#>   student_id full_name      favourite_food   meal_plan
#>   <dbl> <chr>          <chr>            <fct>
#> 1      1 Sunil Huffmann  Strawberry yoghurt Lunch only
#> 2      2 Barclay Lynn    French fries     Lunch only
#> 3      3 Jayendra Lyne   NA               Breakfast and lunch
#> 4      4 Leon Rossini    Anchovies      Lunch only
#> 5      5 Chidiegwu Dunkel Pizza       Breakfast and lunch
#> 6      6 Güvenç Attila   Ice cream      Lunch only
```

Parquet 往往比 RDS 快得多，并且可以在 R 之外使用，但确实需要 `arrow` 包。

## 7.6 数据录入

有时，你需要在 R 脚本中手动组装一个 tibble（数据框），进行一些数据录入。有两个有用的函数可以帮助你完成这项工作，这两个函数的不同之处在于它们是通过列还是通过行来布局 tibble 的。`tibble()` 函数是按列工作的：

```
tibble(
  x = c(1, 2, 5),
  y = c("h", "m", "g"),
  z = c(0.08, 0.83, 0.60)
)
#> # A tibble: 3 x 3
#>   x     y     z
#>   <dbl> <chr> <dbl>
#> 1 1     h     0.08
#> 2 2     m     0.83
#> 3 5     g     0.6
```

按列布局数据可能会难以看到行之间的关系，因此另一种方法是 `tribble()`（`transposed tibble` 的缩写），它允许你逐行布局数据。`tribble()` 是为代码中的数据输入定制的：列标题以 ~ 开头，条目之间用逗号分隔。这使得以易于阅读的形式布局少量数据成为可能：

```
tribble(
  ~x, ~y, ~z,
  1, "h", 0.08,
  2, "m", 0.83,
  5, "g", 0.60
```

```
)  
#> # A tibble: 3 x 3  
#>   x     y     z  
#>   <dbl> <chr> <dbl>  
#> 1     1     h     0.08  
#> 2     2     m     0.83  
#> 3     5     g     0.6
```

## 7.7 小结

在本章中，你学习了如何使用 `read_csv()` 加载 CSV 文件，以及如何使用 `tibble()` 和 `tribble()` 进行自己的数据输入。你了解了 CSV 文件的工作原理，可能会遇到的一些问题，以及如何解决这些问题。在本书中，我们将多次涉及数据导入：从 Excel 和 Google 表格中导入数据的章节 ??，章节 ?? 将向你展示如何从数据库中加载数据，章节 ?? 将介绍如何从 Parquet 文件中加载数据，章节 ?? 将涉及从 JSON 中导入数据，以及 @sec-scraping 将介绍如何从网站上抓取数据。

本书的这个部分即将结束，但还有一个重要的主题需要讨论：如何获取帮助。因此，在下一章中，你将学习一些寻求帮助的好地方，如何创建一个可重现的示例（reprex）以最大化获得良好帮助的机会，以及一些关于跟上 R 世界步伐的一般性建议。

# 8 工作流程：获取帮助

本书并非孤岛，没有任何单一资源能够让你完全掌握 R 语言。当你开始将本书中描述的技术应用于自己的数据时，你很快就会遇到我们没有解答的问题。本节将介绍一些获取帮助和持续学习的技巧。

## 8.1 充分利用互联网

如果你遇到了困难，首先尝试使用 Google 搜索。通常，在查询中加上“R”就能将结果限制在相关的范围内：如果搜索结果没有帮助，那通常意味着没有特定的 R 语言相关结果。此外，添加包名如“tidyverse”或“ggplot2”将有助于将结果缩小到你更熟悉的代码范围内，例如，“如何在 R 中制作箱线图”与“如何在 R 中使用 ggplot2 制作箱线图”。Google 对错误消息特别有用，如果你收到一个错误消息而不知道它是什么意思，试着搜索它！很有可能其他人以前也被这个错误消息搞糊涂过，网上某个地方会有帮助。（如果错误消息不是英文的，运行 `Sys.setenv(LANGUAGE = "en")` 并重新运行代码；你更有可能找到英文错误消息的帮助。）

如果 Google 没有帮助，尝试使用[Stack Overflow](#)。首先花一些时间搜索现有的答案，包括 [R] 标签，以将搜索限制在使用 R 的问题和答案上。

## 8.2 制作 reprexx

如果你的 Google 搜索没有找到有用的信息，制作一个最小化的 reprex (**reproducible example** 的缩写) 是一个很好的主意。一个好的 reprex 会让其他人更容易帮助你，而且通常在制作它的过程中，你自己就会发现问题所在。制作 reprex 有两个部分：

- 首先，你需要让你的代码可复现。这意味着你需要捕获所有内容，即包括任何 `library()` 调用并创建所有必要的对象。确保你已经做到这一点的最简单方法是使用 `reprex` 包。
- 其次，你需要将其简化到最小。去掉所有与问题不直接相关的内容。这通常意味着创建一个比你实际面对的更小、更简单的 R 对象，或者甚至使用内置数据。

这听起来像是一项繁重的工作！确实可能如此，但它的回报非常大

- 80% 的情况下，创建一个出色的 reprex 会揭示问题的根源。令人惊讶的是，多少次在编写一个独立且简化的例子的过程中，你就能够自己解答问题了；
- 另外 20% 的情况下，你已经以易于他人操作的方式捕捉到了问题的本质，这极大地提高了你获得帮助的机会！

在手动创建 reprex 时，很容易不小心遗漏某些内容，导致你的代码无法在别人的计算机上运行。通过使用 reprex 包可以避免这个问题，该包是 tidyverse 的一部分。假设你将这段代码复制到了剪贴板（或者，在 RStudio Server 或 Cloud 上，选择它）：

## 8.2 制作 reprexx

```
y <- 1:4  
mean(y)
```

然后调用 `reprex()`，其默认输出格式为 GitHub 所需的格式：

```
reprex::reprex()
```

如果你正在使用 RStudio，一个精美的 HTML 预览将会在 RStudio 的 Viewer 中显示；否则，它会在你的默认浏览器中显示。`reprex` 会被自动复制到你的剪贴板（在 RStudio Server 或 Cloud 上，你需要手动复制）：

```
``` r  
y <- 1:4  
mean(y)  
#> [1] 2.5  
```
```

这段文本以一种特殊的方式格式化，称为 Markdown，它可以粘贴到像 StackOverflow 或 Github 这样的网站上，并且这些网站会自动将其渲染成代码的样子。以下是这段 Markdown 在 Github 上的渲染效果：

```
y <- 1:4  
mean(y)  
#> [1] 2.5
```

任何人都可以立即复制、粘贴和运行这个例子。

要使你的例子可重复，需要包含三样东西：必要的包、数据和代码。

## 8 工作流程：获取帮助

1. 包应该在脚本的顶部加载，这样很容易看出这个例子需要哪些包。现在是检查你是否在使用每个包的最新版本的好时机；你可能发现了一个自你安装或最后一次更新包以来已经被修复的错误。对于 tidyverse 中的包，最简单的检查方法是运行 `tidyverse_update()`。
2. 包含数据的最简单方法是使用 `dput()` 来生成重新创建它所需的 R 代码。例如，要在 R 中重新创建 `mtcars` 数据集，请按照以下步骤操作：
  1. 在 R 中运行 `dput(mtcars)`
  2. 复制输出结果 t
  3. 在 `reprex` 中，键入 `mtcars <-`，然后粘贴。

尽量使用数据的最小子集，但还要能揭示问题。

3. 花点时间确保你的代码易于他人阅读：
  - 确保你使用了空格，并且你的变量名简洁但富有信息性；
  - 使用注释来指出问题所在；
  - 尽最大努力移除与问题无关的所有内容。

你的代码越短，就越容易理解，也就越容易修复。

最后，通过启动一个新的 R 会话并复制粘贴你的脚本来检查你是否真的创建了一个可重复的例子。

创建 reprexes 并不是一件简单的事情，而且学习创建好的、真正精简的 reprexes 需要一些实践。然而，学习在问题中包含代码，并投入时间使其可重复，将随着你学习和掌握 R 而不断得到回报。

## 8.3 投资自己

你也应该花一些时间在问题发生之前做好解决问题的准备。每天花一点时间学习 R，长期来看会有丰厚的回报。一种方法是关注 tidyverse 团队在[tidyverse 博客](#)上的动态。为了更广泛地了解 R 社区，我们推荐阅读[R Weekly](#)：它是社区的一项努力，每周汇总 R 社区中最有趣的新闻。

## 8.4 小结

本书全貌概览（Whole Game）部分到此结束。现在你已经看到了数据科学过程中最重要的部分：可视化、转换、整理和导入。现在你已经有了一个对整个过程的整体认识，接下来我们将深入到各个小部分的细节。

本书的下一部分，可视化，将深入探讨图形语法和如何使用 ggplot2 创建数据可视化，展示如何使用你迄今为止学到的工具进行探索性数据分析，并介绍创建用于交流的图的良好实践。



## **Part II**

### **可视化**



在阅读了本书的第一部分之后，你（至少从表面上）理解了做数据科学最重要的工具，现在是时候深入细节了。在本书这一部分，你将进一步深入学习数据可视化。

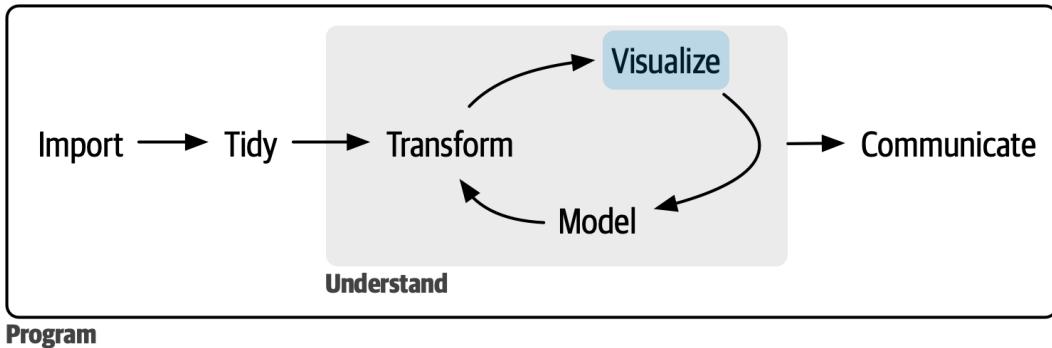


图 8.1: Data visualization is often the first step in data exploration.

每一章都会介绍创建数据可视化的一个或多个方面。

- 在章节 ?? 部分，学习图形的分层语法；
- 在章节 ?? 部分，把可视化与你的好奇心和怀疑精神结合起来，针对数据提出并回答有趣的问题；
- 最后，在章节 ?? 部分，学习如何将探索性图形提升为解释性图形，这些图形有助于新手尽快理解你的分析结果。

这三章带你进入可视化的世界，但还有更多要学习的内容。学习更多知识的绝佳选择是 `ggplot2: ggplot2: Elegant graphics for data analysis`。这本书深入探讨了底层理论，并提供了许多如何将各个部分组合起来解决实际问题的示例。另一个很好的资源是 `ggplot2` 扩展库<https://exts.ggplot2.tidyverse.org/gallery/>。这个网站列出了许多用新的 `geoms` 和 `scales` 扩展 `ggplot2` 的包。如果你试图用 `ggplot2` 做一些看似困难的

事情，这是一个很好的出发点。

# 9 图层

## 9.1 引言

在章节 `??` 中，你学到的远不止如何制作散点图、条形图和箱线图。你学习了利用 `ggplot2` 可以制作任何类型图形的基础知识。

在本章中你将在这个基础上进行扩展，学习图形的分层语法。首先，我们将更深入地探讨美学映射、几何对象和切面。然后，你将了解 `ggplot2` 在创建图形时背后进行的统计转换。这些转换用于计算新的绘图值，例如条形图中的条形高度或箱线图的中位数。你还将学习位置调整，这会修改几何对象在图表中的显示方式。最后，我们将简要介绍坐标系。

我们不会涵盖这些图层中的每一个函数和选项，但我们将引导你了解 `ggplot2` 提供的最重要和最常用的功能，并向你介绍扩展 `ggplot2` 的包。

### 9.1.1 必要条件

本章重点介绍 `ggplot2`。要访问本章中使用的数据集、帮助页面和函数，请运行以下代码加载 `tidyverse`:

```
library(tidyverse)
```

## 9.2 美学映射

“图片的最大价值在于它能够迫使我们注意到我们从未预期到的内容。”

— John Tukey

请记住，ggplot2 包附带的 mpg 数据框包含了关于 38 种不同汽车型号的 234 条观测数据。

```
mpg
#> # A tibble: 234 x 11
#>   manufacturer model displ year cyl trans     drv     cty     hwy fl
#>   <chr>        <chr>  <dbl> <int> <int> <chr>    <chr> <int> <int> <chr>
#> 1 audi         a4      1.8  1999     4 auto(15) f       18     29 p
#> 2 audi         a4      1.8  1999     4 manual(m5) f      21     29 p
#> 3 audi         a4      2     2008     4 manual(m6) f      20     31 p
#> 4 audi         a4      2     2008     4 auto(av)   f      21     30 p
#> 5 audi         a4      2.8  1999     6 auto(15) f      16     26 p
#> 6 audi         a4      2.8  1999     6 manual(m5) f     18     26 p
#> # i 228 more rows
#> # i 1 more variable: class <chr>
```

mpg 中的变量包括：

1. **displ**: 汽车发动机的排量，单位为升。这是一个数值型变量。

## 9.2 美学映射

2. **hwy**: 高速公路上汽车的燃油效率，以每加仑行驶的英里数（mpg）为单位。当两辆汽车行驶相同的距离时，燃油效率低的汽车比燃油效率高的汽车消耗更多的燃油。这是一个数值型变量。
3. **class**: 汽车的型号。这是一个分类变量。

让我们首先通过可视化不同车型类别（**class**）的排量（**displ**）和高速公路燃油效率（**hwy**）之间的关系来开始。我们可以使用散点图来实现这一点，其中数值变量映射到 **x** 和 **y** 美学上，而分类变量则映射到 **color** 或 **shape** 等美学上。

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

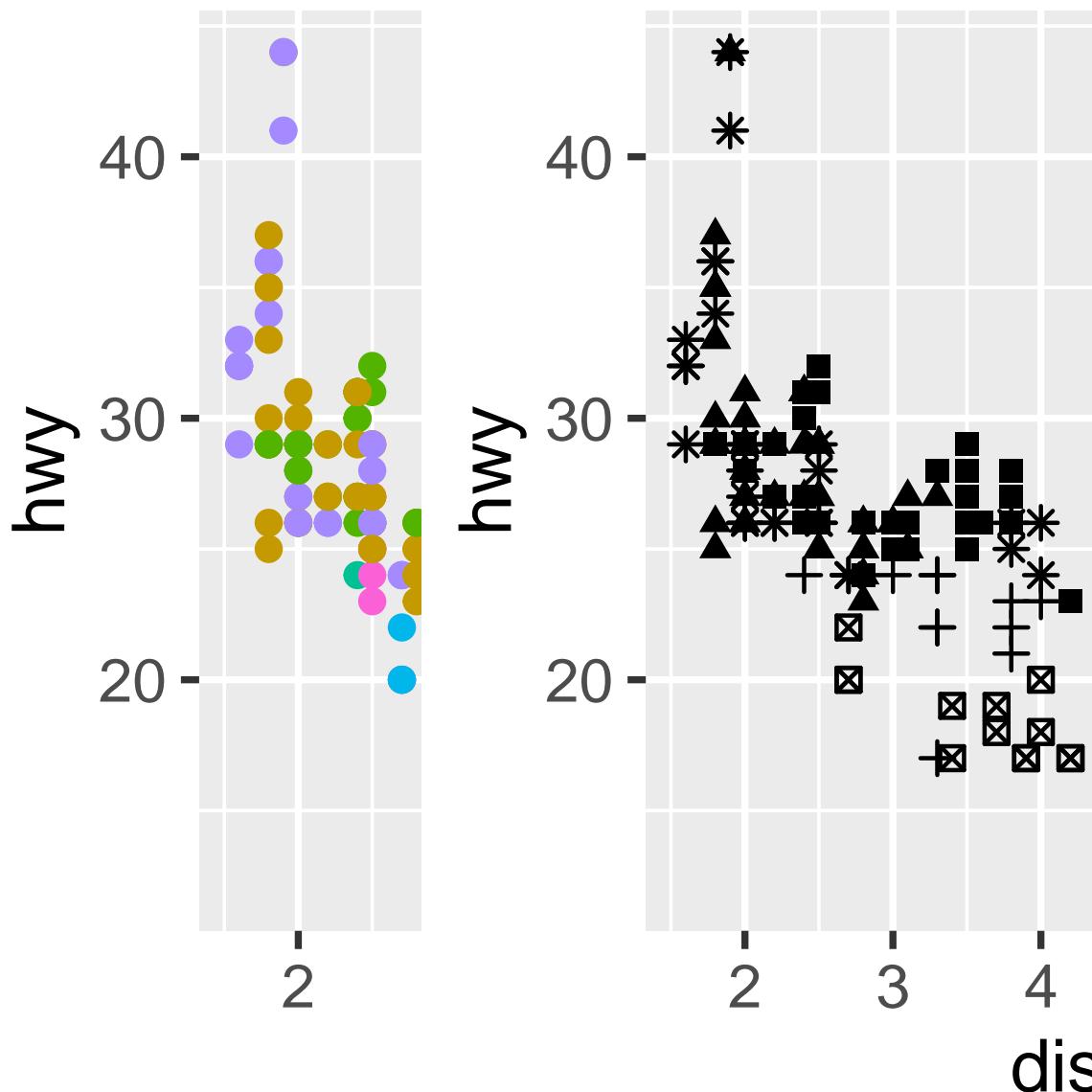
# Right
ggplot(mpg, aes(x = displ, y = hwy, shape = class)) +
  geom_point()

#> Warning: The shape palette can deal with a maximum of 6 discrete values because more
#> than 6 becomes difficult to discriminate
#> i you have requested 7 values. Consider specifying shapes manually if you
#> need that many have them.
#> Warning: Removed 62 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```

当类别（**class**）映射到形状（**shape**）时，我们得到了两个警告：

- 1: 形状调色板最多只能处理 6 个离散值，因为超过 6 个值后就很难区分；但你有 7 个。如果你必须使用它们，请考虑手动指定形状。

9 图层



## 9.2 美学映射

2: 删除了包含缺失值的 62 行 (`geom_point()`)。

由于 `ggplot2` 默认一次只会使用六个形状，因此当使用形状美学时，额外的组将不会被绘制。第二个警告与此相关——数据集中有 62 辆 SUV，它们没有被绘制出来。

类似地，我们也可以将类别 (`class`) 映射到大小 (`size`) 或透明度 (`alpha`) 美学上，它们分别控制点的大小和透明度。

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

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

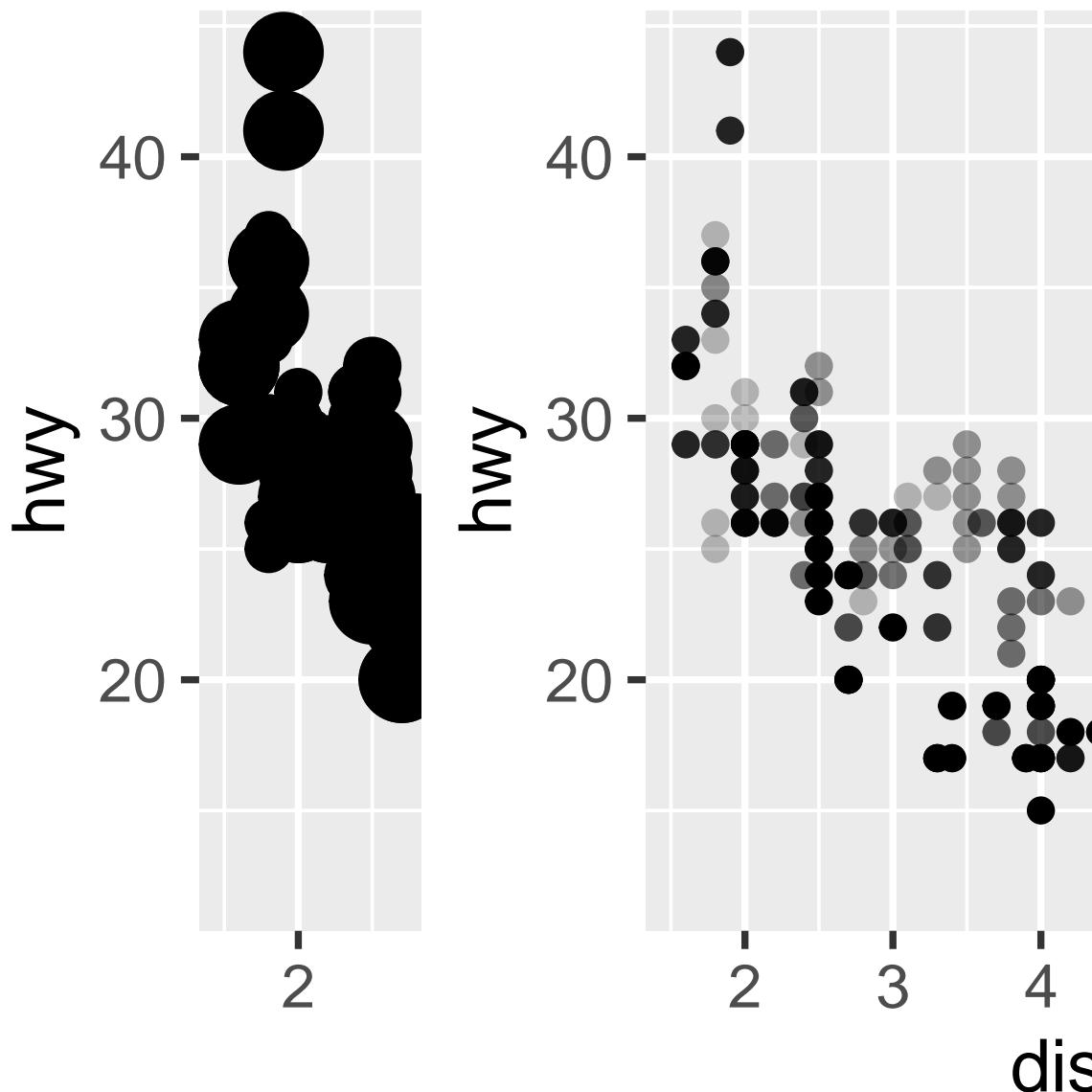
这两种方法都会产生警告：

不建议将 `alpha` 用作离散变量。

将一个无序的离散（分类）变量（如车型 `class`）映射到一个有序的美学属性（如大小 `size` 或透明度 `alpha`）通常不是明智之举，因为它暗含了一个实际上并不存在的排名。

一旦你映射了一个美学属性，`ggplot2` 就会处理其余部分。它会选择一个合理的刻度来与这个美学属性一起使用，并构造一个图例来解释水平和值之间的映射关系。对于 `x` 和 `y` 美学属性，`ggplot2` 不会创建图例，但会创建一个带有

9 图层



230

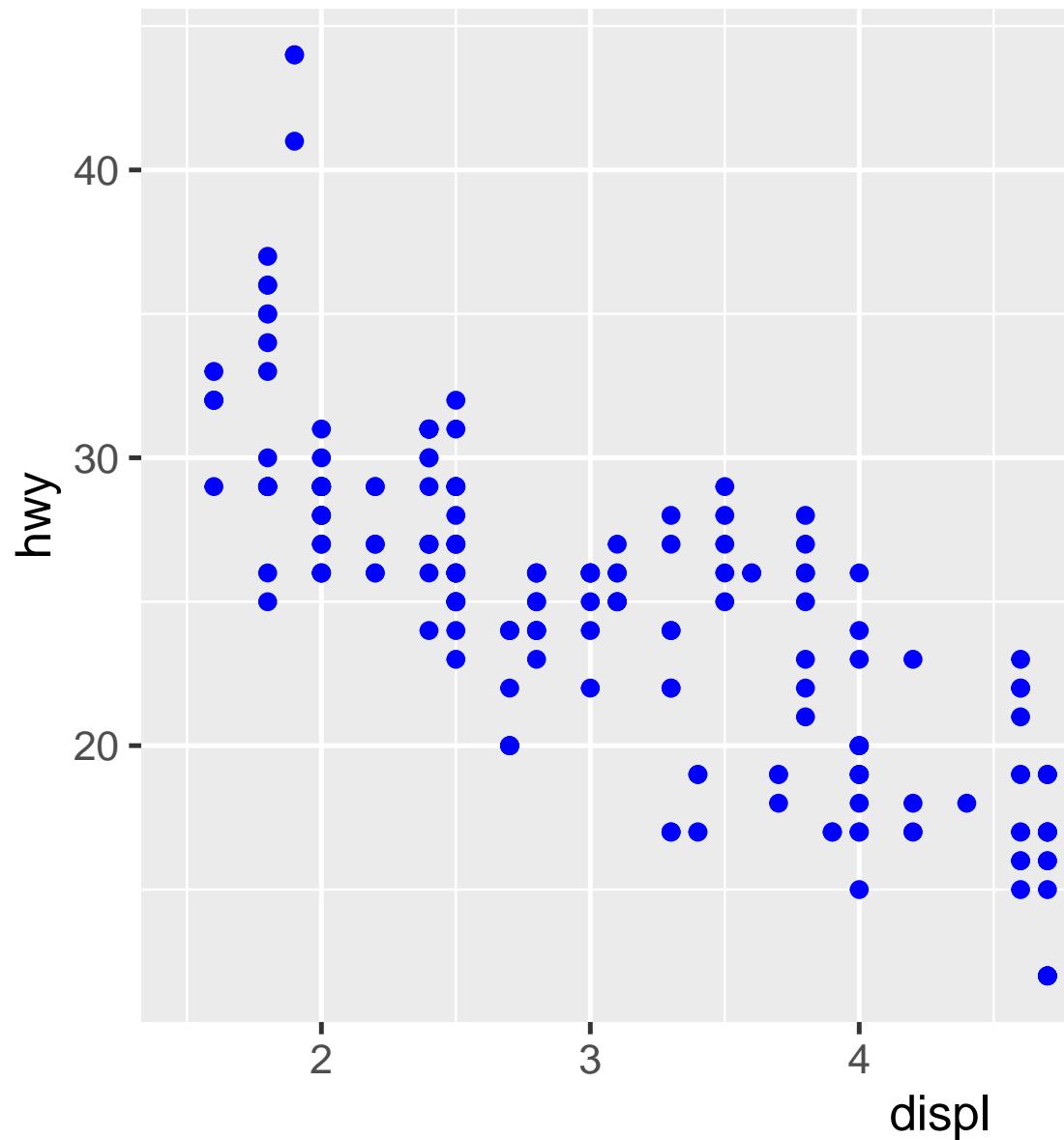
## 9.2 美学映射

刻度线和标签的坐标轴线。坐标轴线提供了与图例相同的信息；它解释了位置和值之间的映射关系。

你也可以通过几何对象函数的参数（在 `aes()` 之外）来手动设置你的几何对象的视觉属性，而不是依赖变量映射来确定外观。例如，我们可以让我们的图形中的所有点都呈现蓝色：

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

9 图层



## 9.2 美学映射

在这里颜色并不传达关于变量的信息，而只是改变图的外观。你需要选择一个对该美学属性有意义的值：

- 颜色的名称作为字符串，例如 `color = "blue"`
- 点的大小以毫米为单位，例如 `size = 1`
- 点的形状作为一个数字，例如 `shape = 1`, 如图图 ?? 所示。

到目前为止，我们讨论了在使用点几何对象（point geom）创建散点图时可以映射或设置的美学属性。你可以在<https://ggplot2.tidyverse.org/articles/ggplot2-specs.html> 的美学规范文档中了解更多关于所有可能的美学映射的信息。

你可以在一个图中使用的具体美学属性取决于你用来表示数据的几何对象（geom）。在下一节中，我们将更深入地探讨几何对象。

### 9.2.1 练习

1. 创建一个 `hwy` 和 `displ` 的散点图，其中的点是粉红色填充的三角形。
2. 以下代码为什么没有生成带有蓝色点的图：

```
ggplot(mpg) +  
  geom_point(aes(x = displ, y = hwy, color = "blue"))
```
3. 描边美学（stroke aesthetic）是做什么用的？它与哪些形状一起工作？（提示：使用 `?geom_point`）
4. 如果你将美学映射到变量名之外的其他东西，比如 `aes(color = displ < 5)`，会发生什么？注意，你还需要指定 `x` 和 `y`。

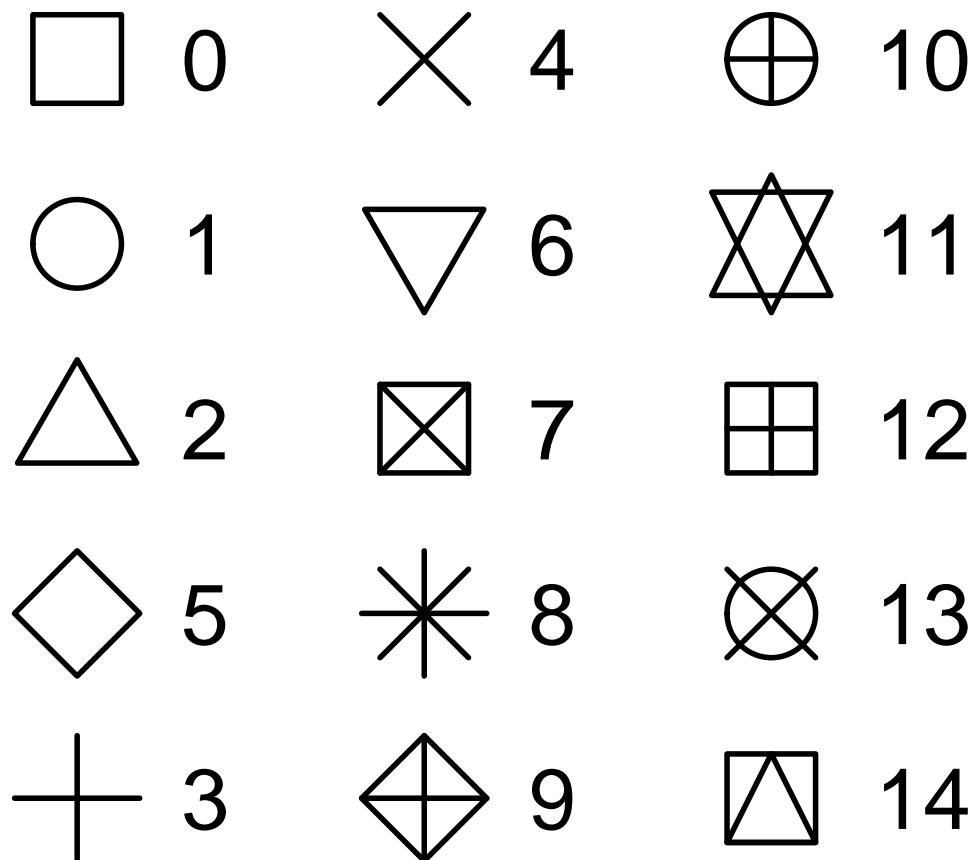


图 9.1: R has 25 built-in shapes that are identified by numbers. There are

234

some seeming duplicates: for example, 0, 15, and 22 are all squares. The difference comes from the interaction of the `color` and `fill` aesthetics. The hollow shapes (0–14) have a border determined by `color`; the solid shapes (15–20) are filled with `color`; the filled shapes (21–24) have a border of `color` and are filled with `fill`. Shapes are arranged to keep similar shapes next to each other.

## 9.3 几何对象

下面两张图有什么相似之处？

两张图包含相同的 x 变量和 y 变量，并且都描述了相同的数据。但是，这两张图并不完全相同。每张图都使用不同的几何对象（geom）来表示数据。左侧的图使用了点几何对象（point geom），而右侧的图表使用了平滑几何对象（smooth geom），即用平滑线拟合到数据上。

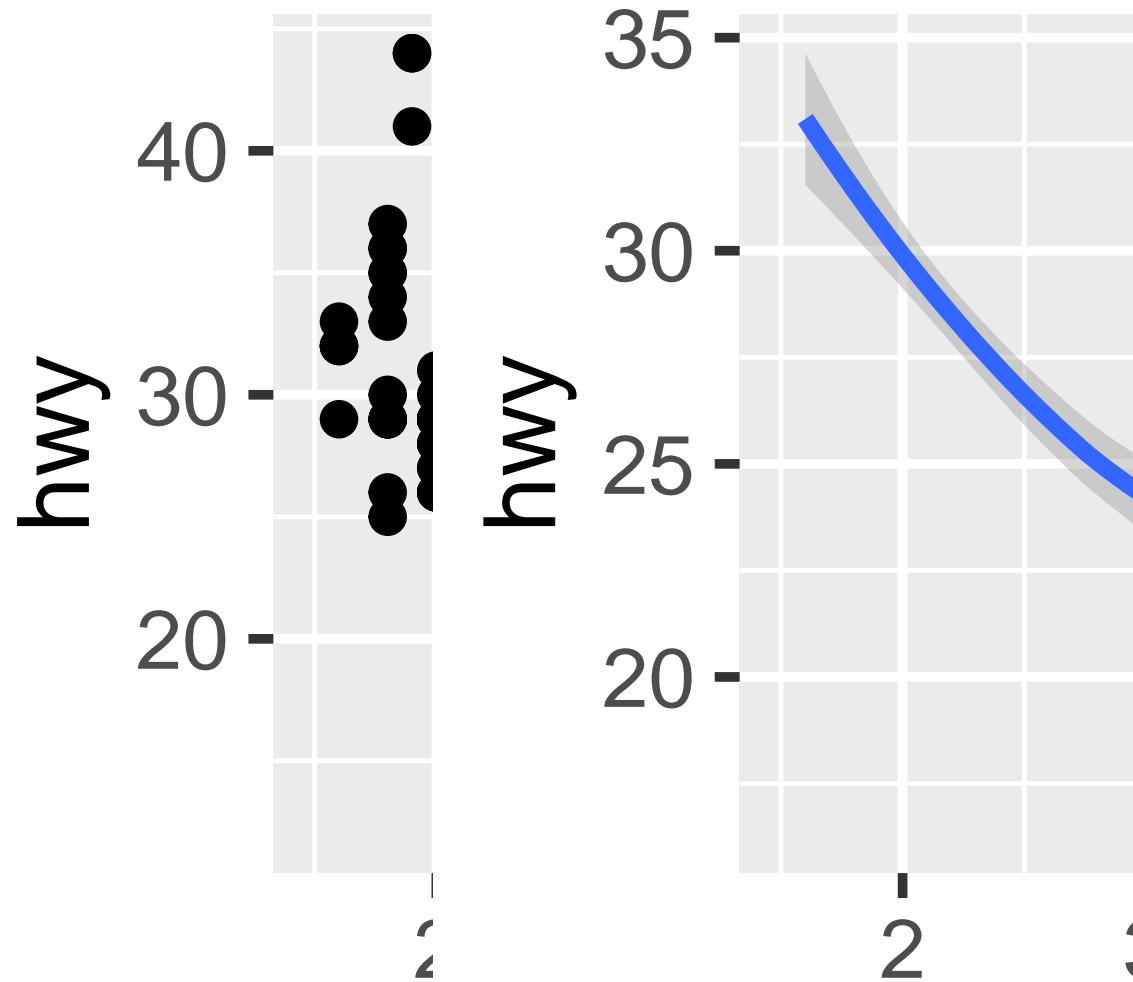
要在你的图中更改几何对象（geom），请更改你添加到 `ggplot()` 函数中的几何对象函数。例如，要绘制上述图可以使用以下代码：

```
# Left
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point()

# Right
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

在 ggplot2 中，每个 geom 函数都接受一个映射参数，这个参数可以在 geom 图层中本地定义，也可以在 `ggplot()` 图层中全局定义。然而，并不是每个美学属性（aesthetic）都适用于每个 geom。你可以设置点的形状，但你不能设置线的“形状”。如果你尝试这样做，ggplot2 会静默地忽略该美学映射。另一方面，你可以设置线的线型。`geom_smooth()` 会根据你映射到线型变量的不同值，绘制具有不同线型的平滑线。

9 图层



### 9.3 几何对象

```
# Left
ggplot(mpg, aes(x = displ, y = hwy, shape = drv)) +
  geom_smooth()

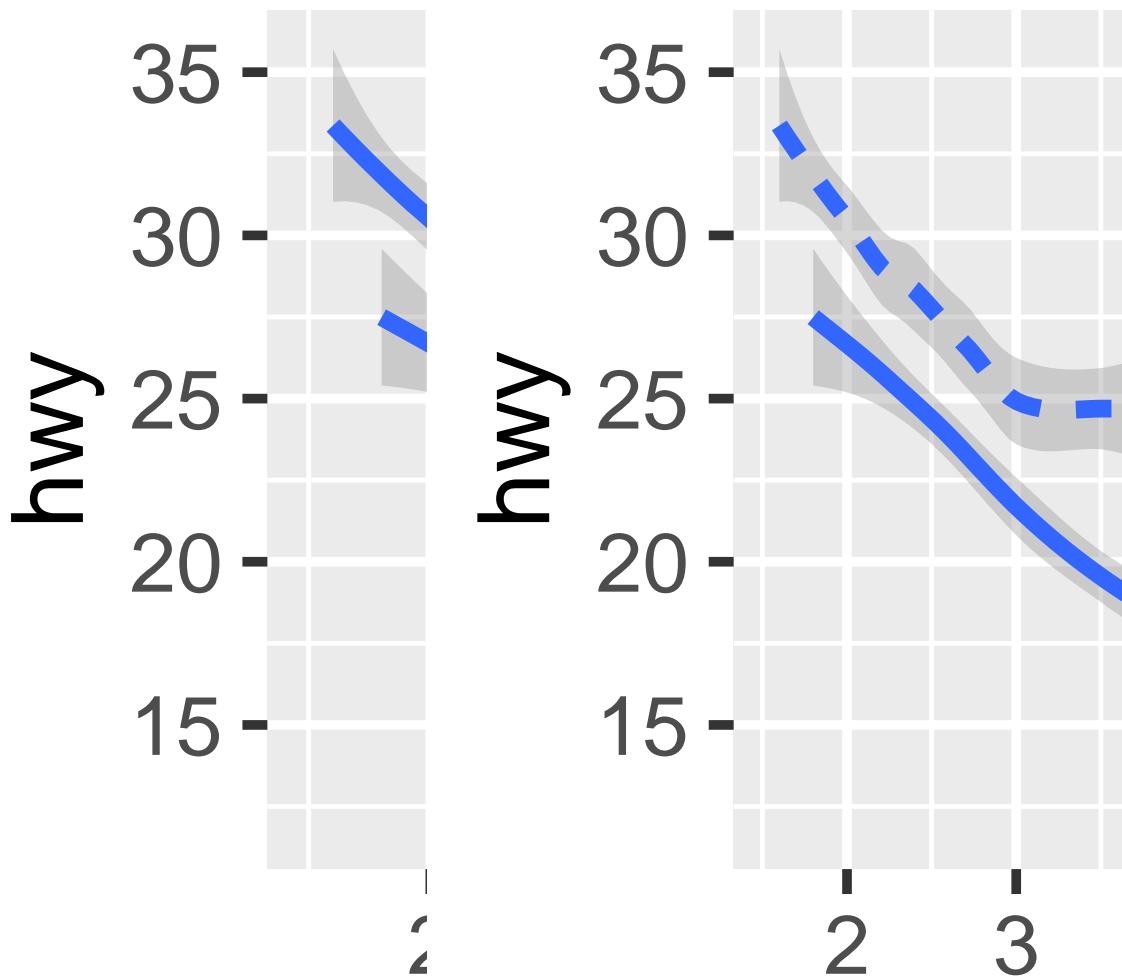
# Right
ggplot(mpg, aes(x = displ, y = hwy, linetype = drv)) +
  geom_smooth()
```

在这里，`geom_smooth()` 根据汽车的 `drv` 值（描述了汽车的驱动方式）将汽车数据分成三条线。一条线描述了所有值为 4 的点，一条线描述了所有值为 `f` 的点，还有一条线描述了所有值为 `r` 的点。在这里，4 代表四轮驱动，`f` 代表前轮驱动，`r` 代表后轮驱动。

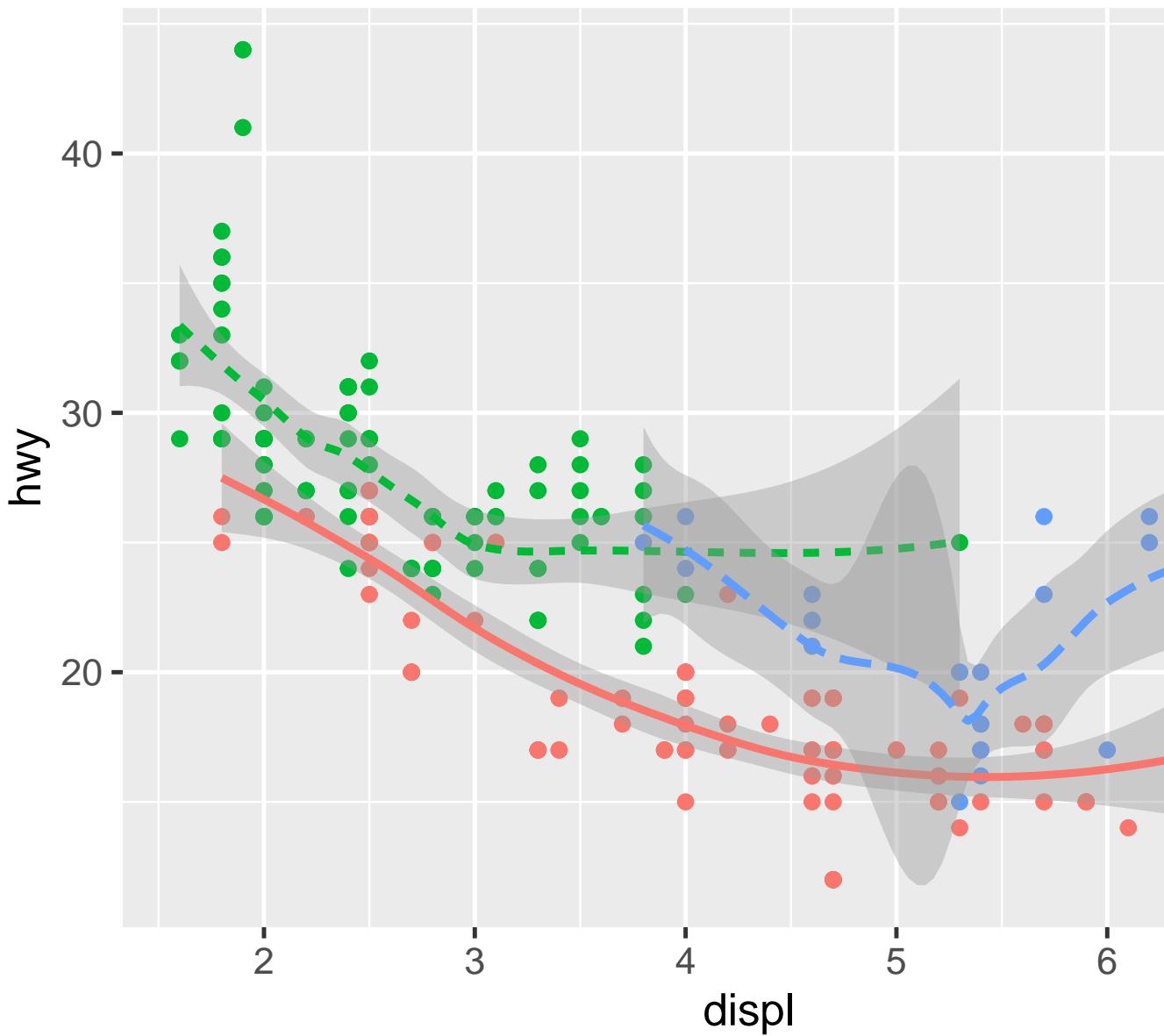
如果这听起来有些奇怪，我们可以通过将这三条线叠加在原始数据上，并根据 `drv` 的值给所有内容上色，来使其更加清晰。

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

9 图层



### 9.3 几何对象



## 9 图层

请注意，这个图在同一个图形中包含了两个几何对象（geoms）。

许多几何对象（如 `geom_smooth()`）使用单个几何对象来显示多行数据。对于这些几何对象，你可以将组（group）美学属性设置为分类变量来绘制多个对象。ggplot2 会为分组变量的每个值绘制一个独立的对象。在实践中，当你将美学属性映射到离散变量时（如线型示例），ggplot2 会自动为这些几何对象分组数据。依赖这个特性是很方便的，因为组（group）美学属性本身不会给几何对象添加图例或区分特征。

```
# Left
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()

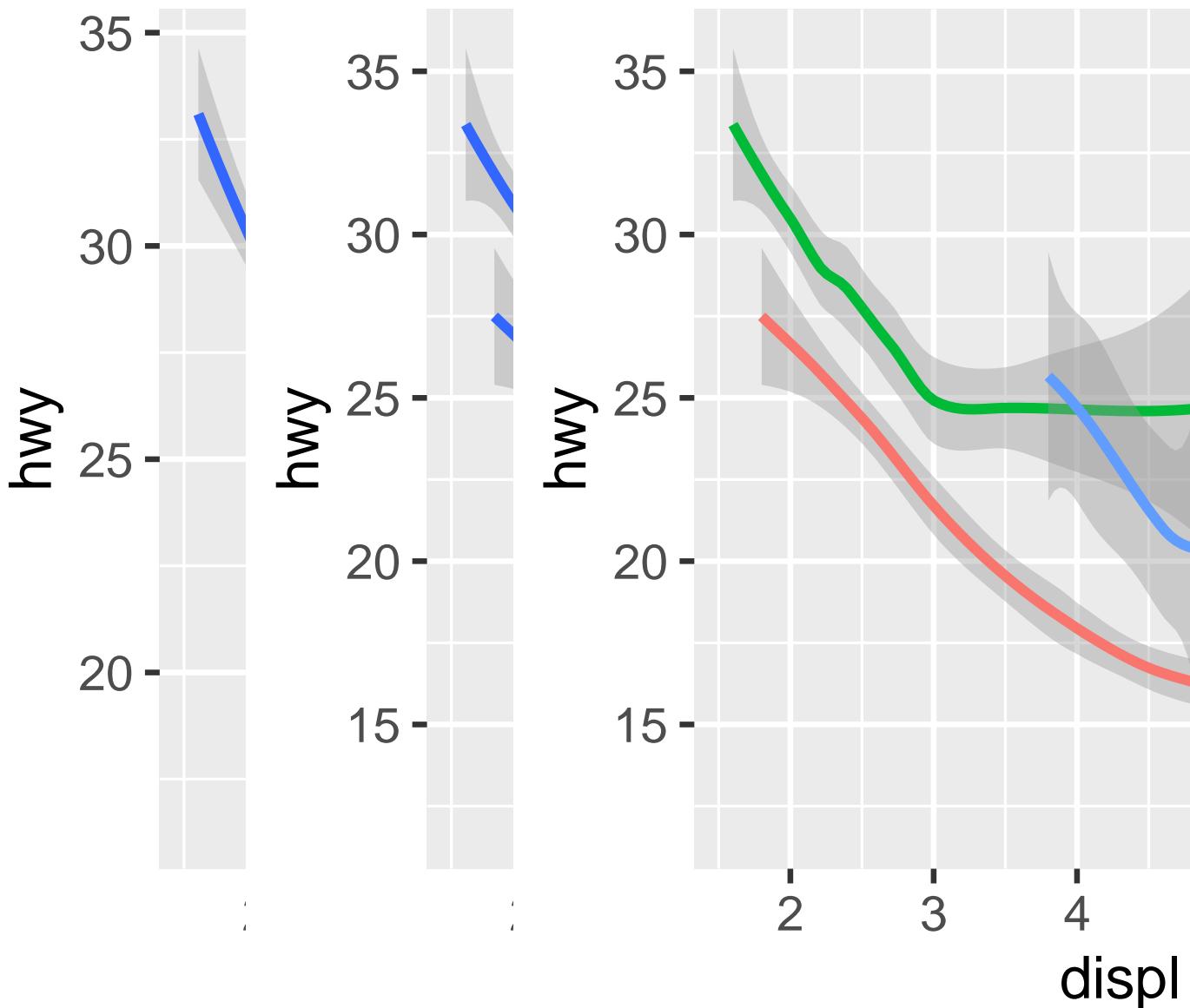
# Middle
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(group = drv))

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

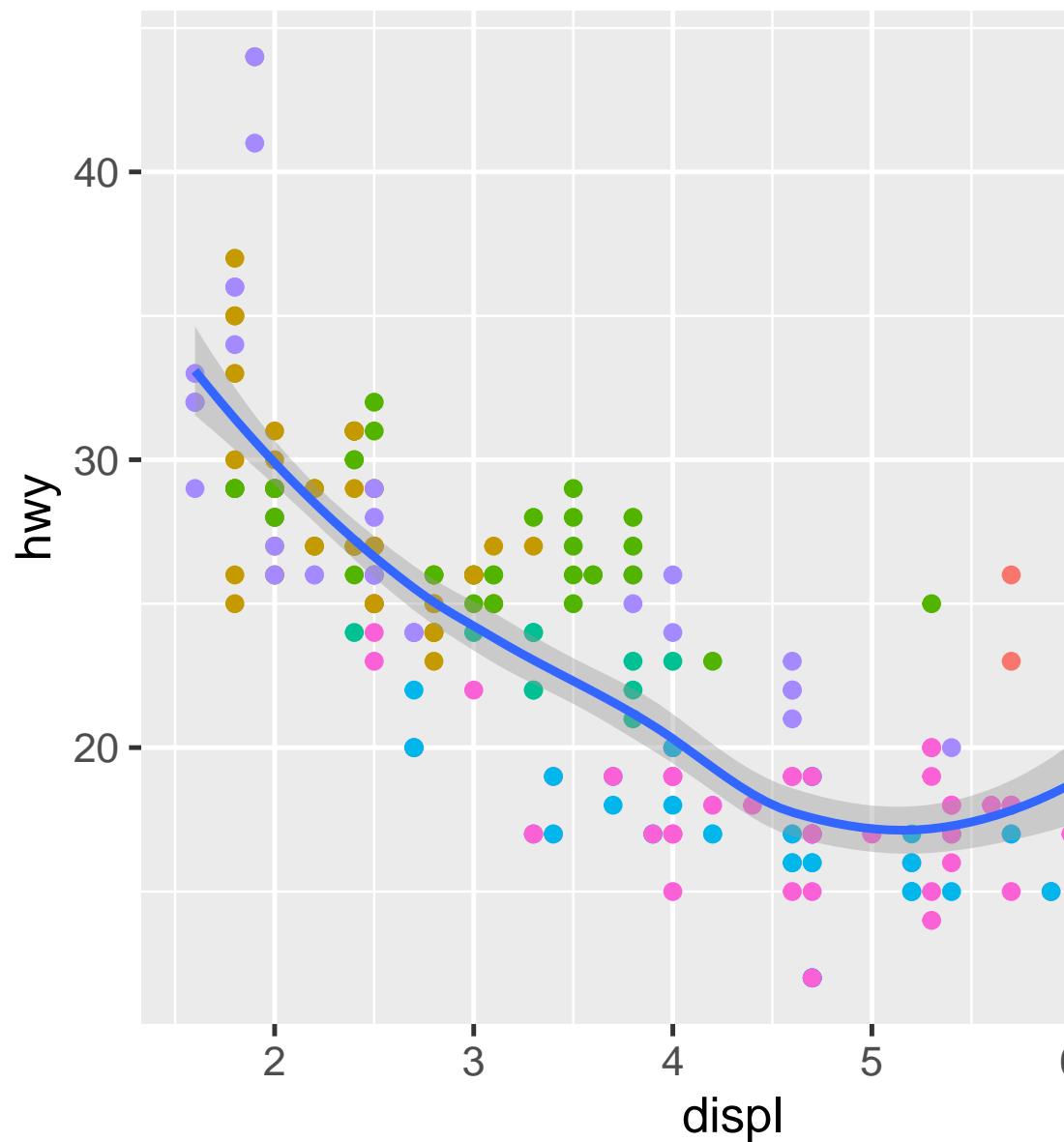
如果你在 `geom` 函数中放置映射，ggplot2 会将这些映射视为该图层的本地映射。它将使用这些映射来扩展或覆盖该图层的全局映射。这使得在不同的图层中显示不同的美学属性成为可能。

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

### 9.3 几何对象



9 图层

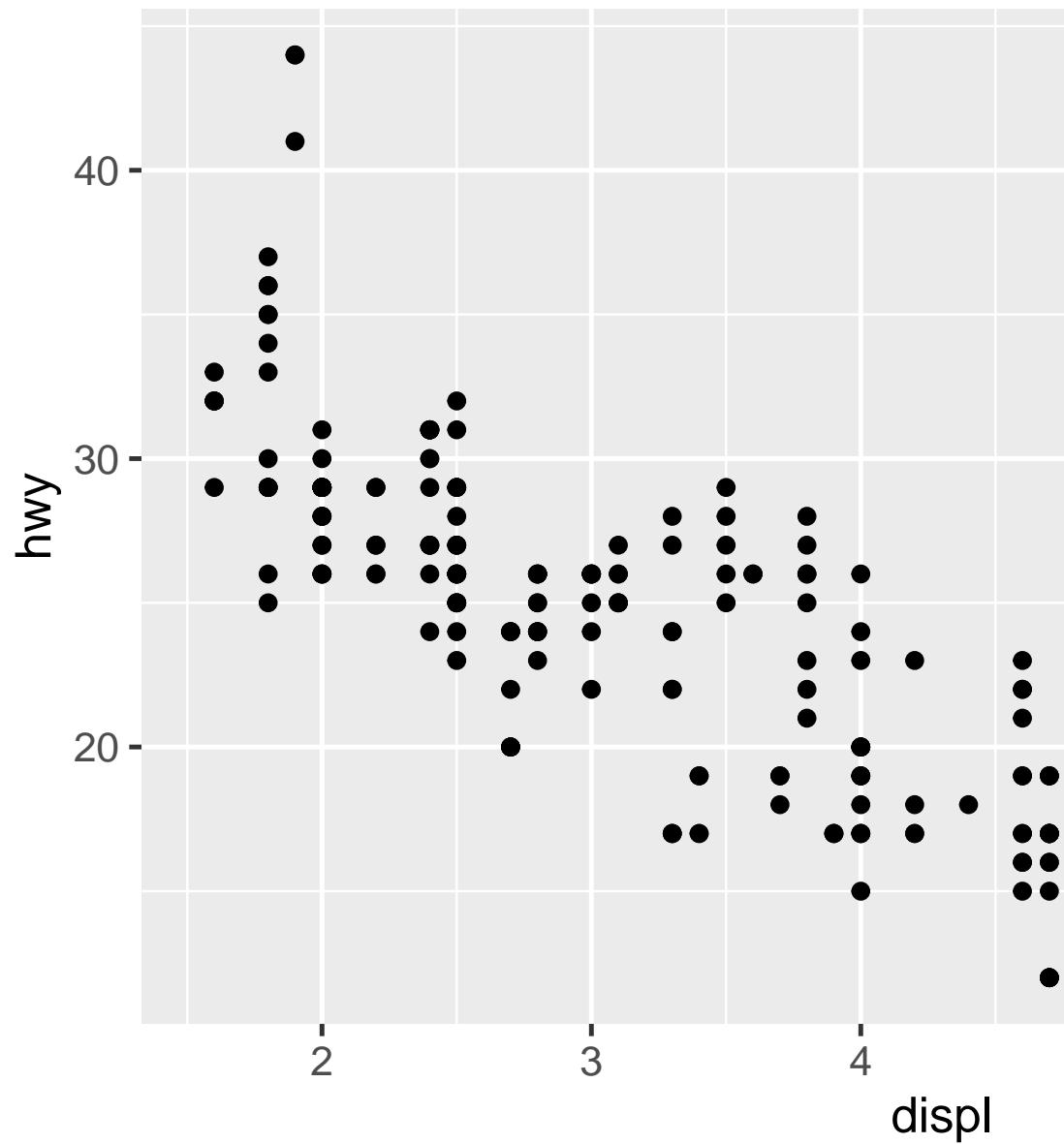


### 9.3 几何对象

你可以使用相同的思路来为每一层指定不同的数据。在这里，我们使用红色的点和空心的圆圈来突出显示两座车。`geom_point()` 中的局部数据参数 (local data argument) 仅针对那一层覆盖了 `ggplot()` 中的全局数据参数 (global data argument)。

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_point(
    data = mpg |> filter(class == "2seater"),
    color = "red"
  ) +
  geom_point(
    data = mpg |> filter(class == "2seater"),
    shape = "circle open", size = 3, color = "red"
  )
```

9 图层



### 9.3 几何对象

几何对象（`geoms`）是 `ggplot2` 的基本构建模块。你可以通过改变图形的 `geom` 来完全改变其外观，而不同的 `geoms` 可以揭示数据的不同特征。例如下面的直方图和密度图揭示了高速公路里程的分布是双峰的且向右偏斜，而箱线图则揭示了两个潜在的异常值。

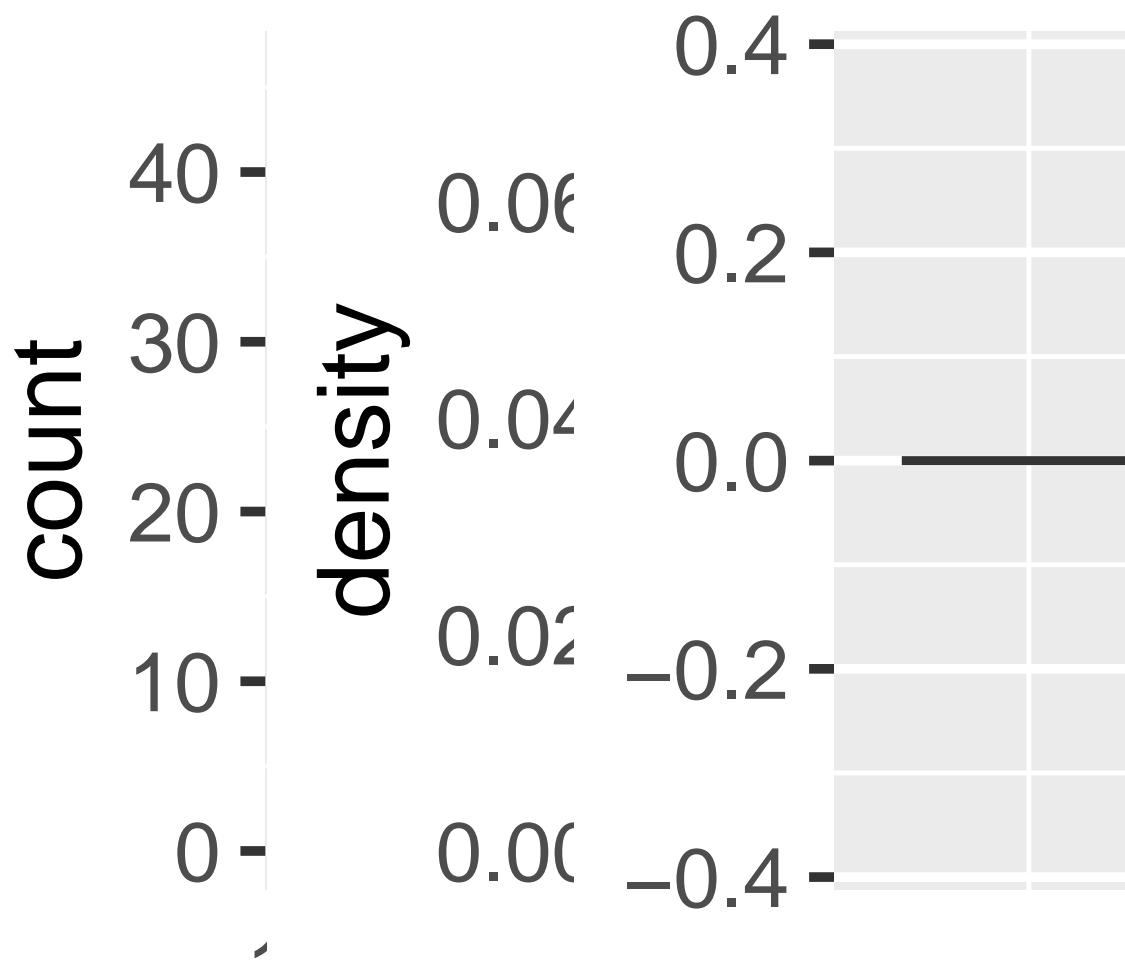
```
# Left
ggplot(mpg, aes(x = hwy)) +
  geom_histogram(binwidth = 2)

# Middle
ggplot(mpg, aes(x = hwy)) +
  geom_density()

# Right
ggplot(mpg, aes(x = hwy)) +
  geom_boxplot()
```

`ggplot2` 提供了超过 40 个 `geoms`，但这并不覆盖所有可能创建的图形。如果你需要一个不同的 `geom`，我们建议先查看扩展包，看看是否有人已经实现了它（可以在<https://exts.ggplot2.tidyverse.org/gallery/>找到一个示例）。例如，`ggridges` 包 (<https://wilkelab.org/ggridges>) 对于制作山脊线图 (ridgeline plots) 很有用，这种图可以用于可视化数值变量在不同类别变量水平下的密度。在下面的图形中，我们不仅使用了一个新的 `geom` (`geom_density_ridges()`)，而且我们还将相同的变量映射到多个美学属性 (`drv` 到 `y`、`fill` 和 `color`)，并设置了一个美学属性 (`alpha = 0.5`) 以便密度曲线透明。

9 图层

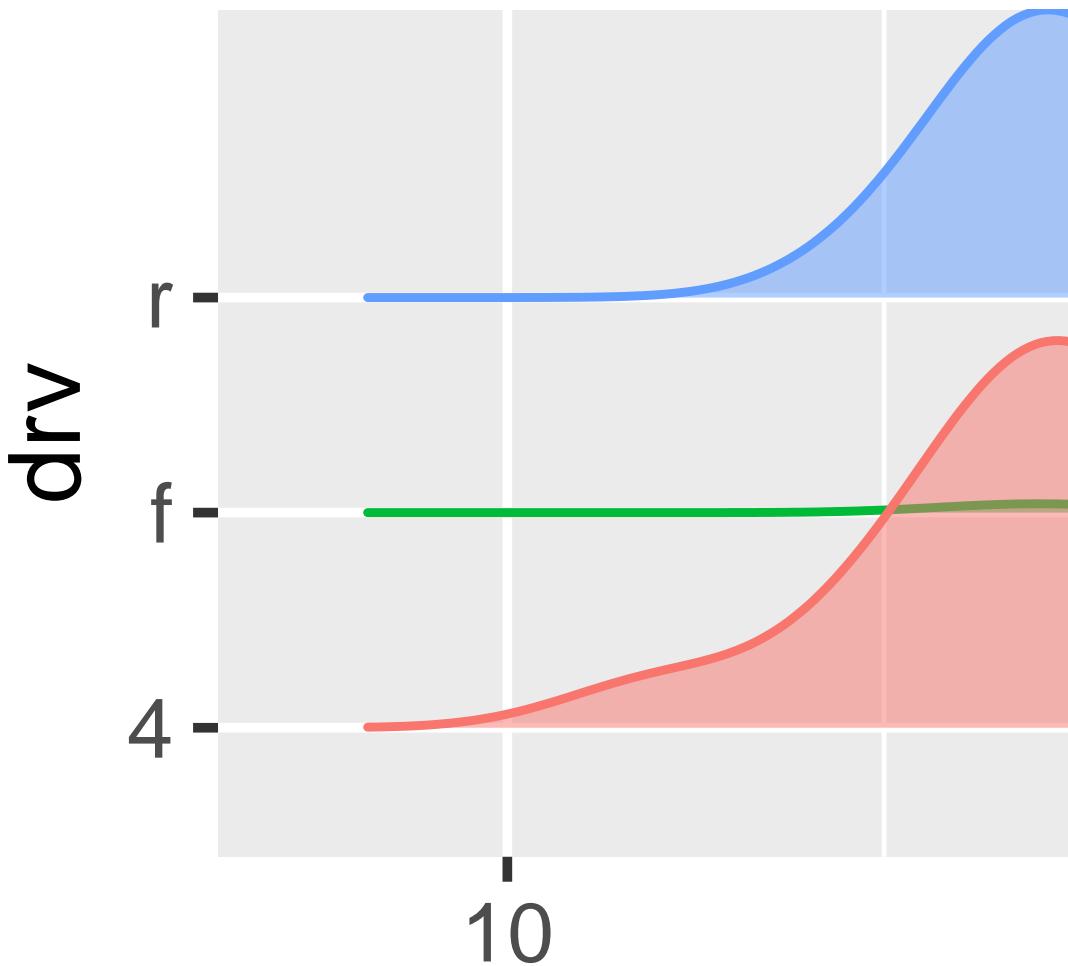


### 9.3 几何对象

```
library(ggridges)

ggplot(mpg, aes(x = hwy, y = drv, fill = drv, color = drv)) +
  geom_density_ridges(alpha = 0.5, show.legend = FALSE)
#> Picking joint bandwidth of 1.28
```

9 图层



## 9.4 分面

要了解 `ggplot2` 提供的所有 `geoms` 的全面概述以及包中的所有函数，最好的地方是参考<https://ggplot2.tidyverse.org/reference>。要深入了解任何单一的 `geom`，请使用帮助函数（例如 `?geom_smooth`）。

### 9.3.1 练习

1. 你会使用哪种 `geom` 来绘制折线图？箱线图？直方图？面积图？
2. 在本章的前面部分，我们使用了 `show.legend` 而没有解释它：

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

`show.legend = FALSE` 在这里的作用是什么？如果你把它去掉会怎么样？你觉得我们之前为什么用它？

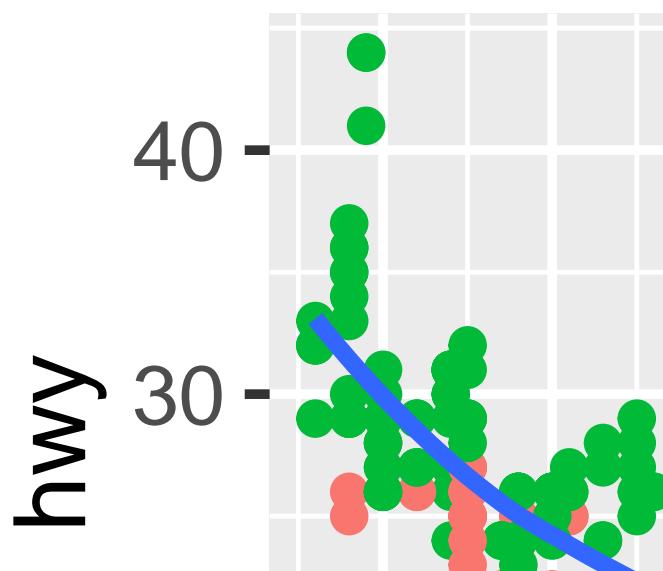
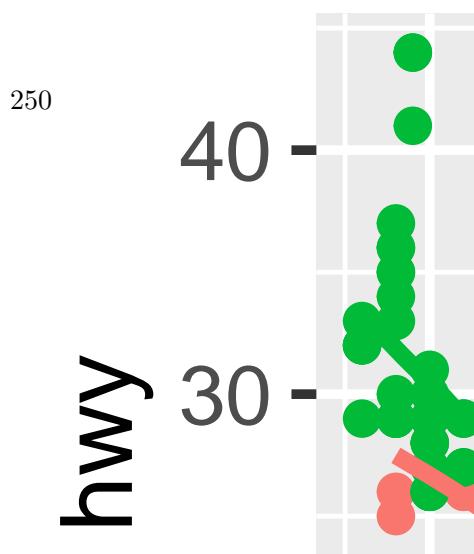
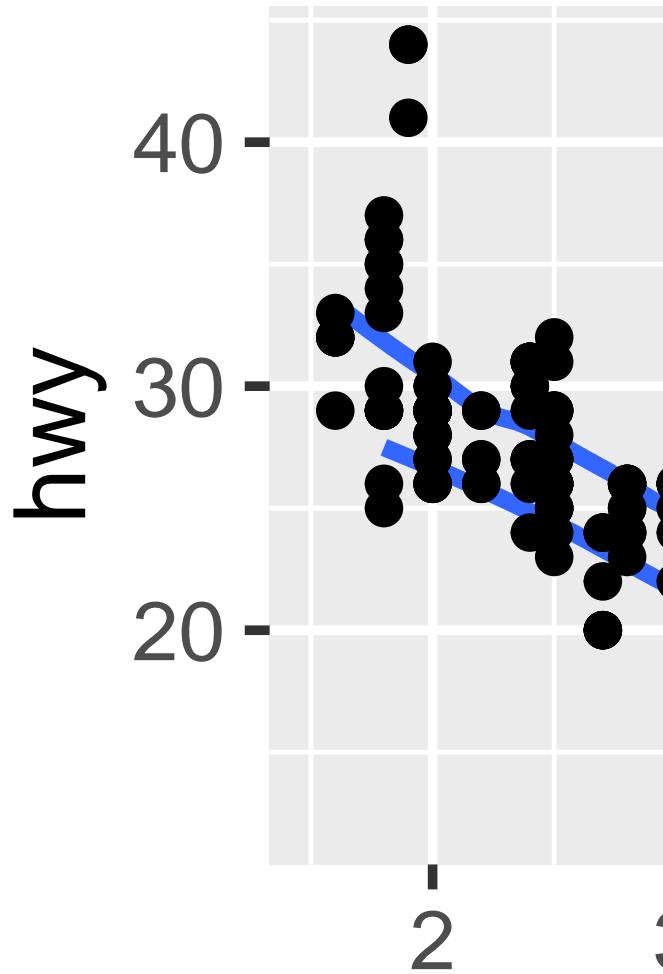
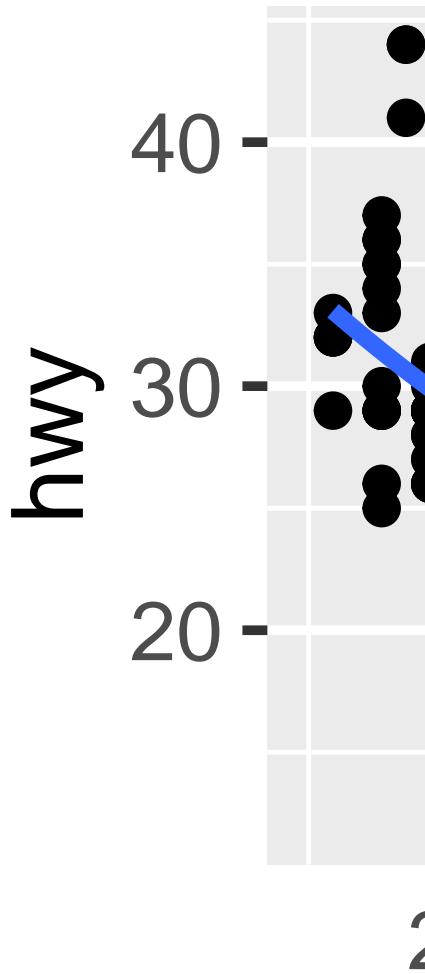
3. `geom_smooth()` 的参数 `se` 起什么作用？
4. 重新创建生成以下图形所需的 R 代码。请注意，图中用到的分类变量的都是 `drv`。

## 9.4 分面

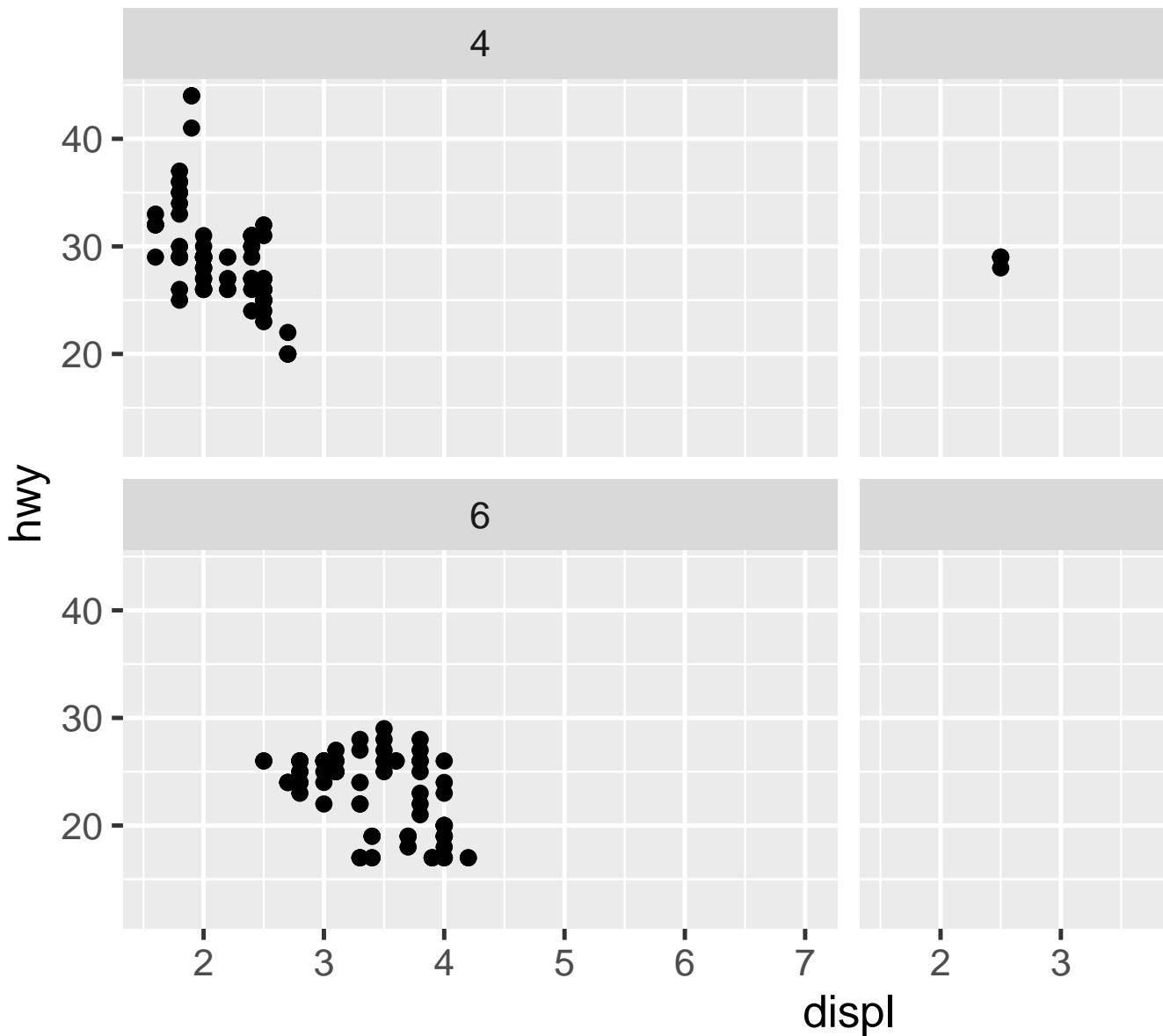
在章节 `??`，你学习了使用 `facet_wrap()` 进行分面，该方法可以将一个图形分割成子图，每个子图都基于一个分类变量显示数据的一个子集。

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

9 图层



9.4 分面

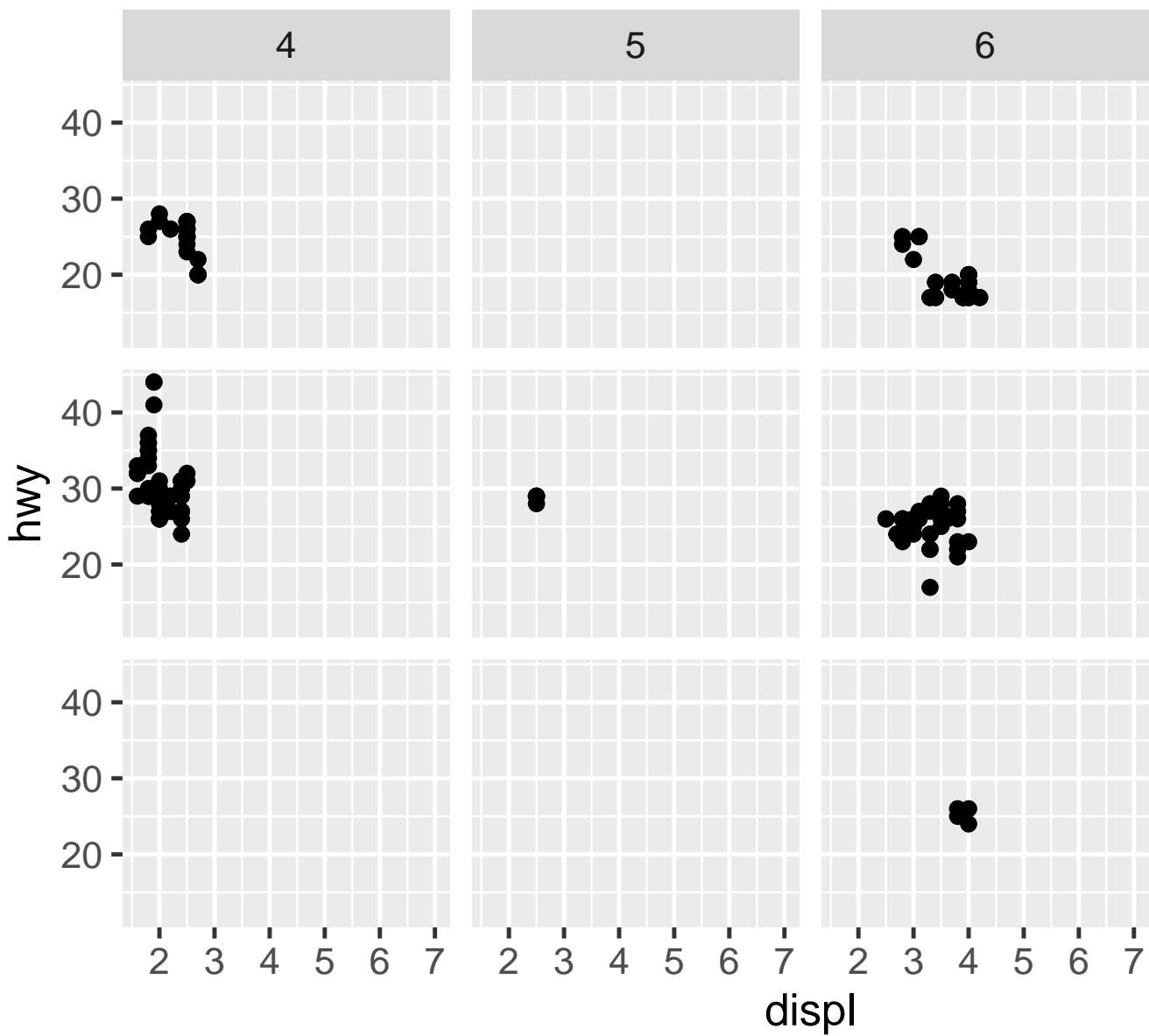


## 9 图层

要使用两个变量的组合来对你的图形进行分面，你需要从 `facet_wrap()` 切换到 `facet_grid()`。`facet_grid()` 的第一个参数也是一个公式，但现在它是一个双面公式：`rows ~ cols`。

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

9.4 分面

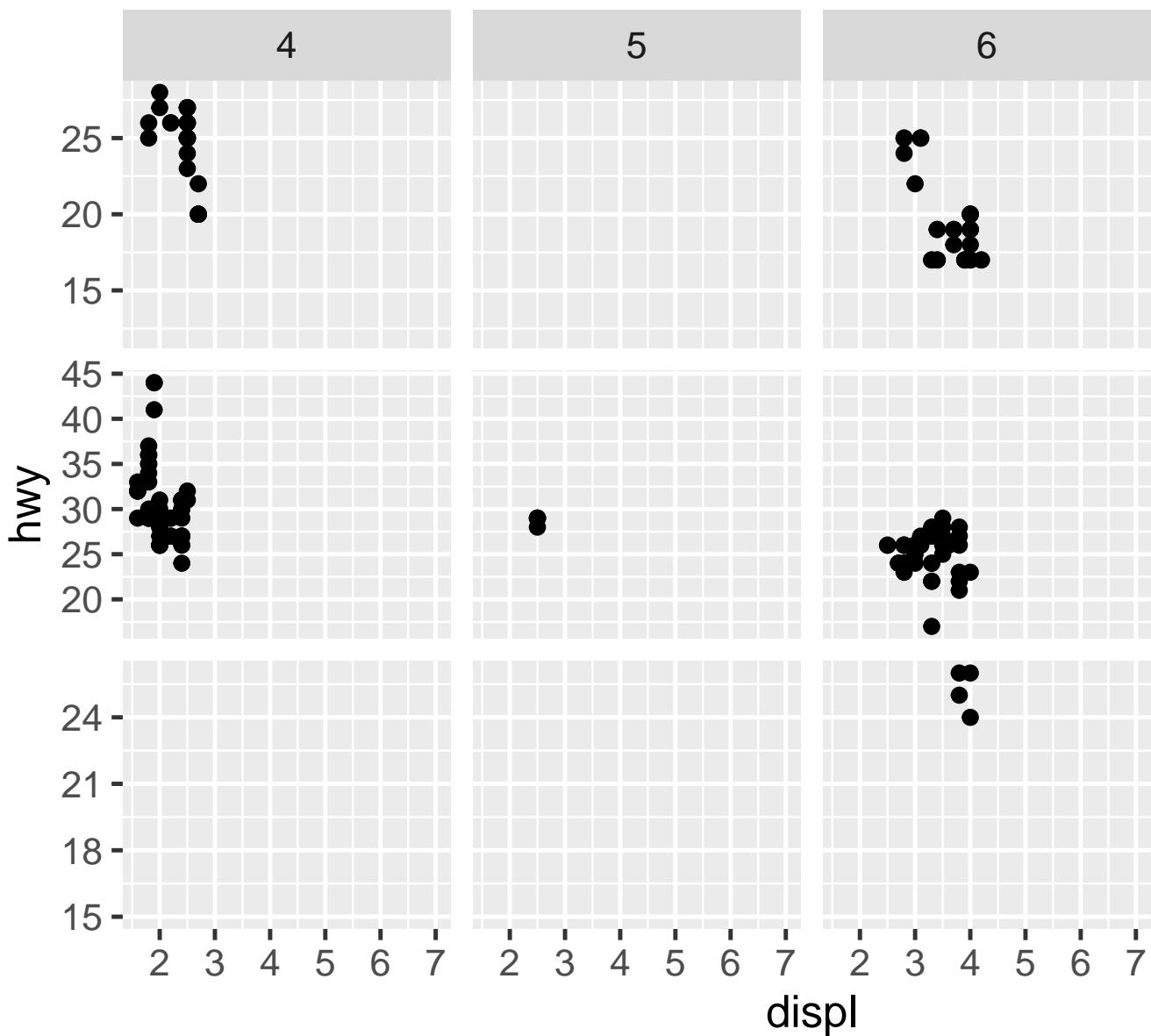


## 9 图层

默认情况下，每个分面共享相同的 x 轴和 y 轴的刻度和范围。当你想跨分面比较数据时，这很有用，但当你希望更好地可视化每个分面内的关系时，这可能会有所限制。将分面函数中的 `scales` 参数设置为"free" 将允许行和列之间使用不同的轴刻度，“`free_x`” 将允许行之间使用不同的刻度，“`free_y`” 将允许列之间使用不同的刻度。

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(drv ~ cyl, scales = "free_y")
```

9.4 分面



### 9.4.1 练习

- 如果你对一个连续变量进行分面会发生什么?
- 上面的图中使用 `facet_grid(drv ~ cyl)` 生成的空单元格是什么意思?  
运行以下代码，它们与生成的图有什么关系?

```
ggplot(mpg) +
  geom_point(aes(x = drv, y = cyl))
```

- 以下代码生成了什么样的图形? . 在这里代表什么?

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

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

- 查看本节中的第一个分面图:

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

使用分面 (faceting) 而不是颜色美学 (color aesthetic) 的优势是什么?  
劣势是什么? 如果你有一个更大的数据集, 这种平衡可能会如何改变?

- 阅读`?facet_wrap`的帮助文档, `nrow`是做什么的? `ncol`呢? 还有哪些选项控制单个面板的布局? 为什么 `facet_grid()` 没有 `nrow` 和 `ncol` 参数?

## 9.5 统计变换

6. 下面的哪个图使得比较不同驱动方式的汽车的发动机排量 (`displ`) 更容易?

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(drv ~ .)

ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(. ~ drv)
```

7. 使用 `facet_wrap()` 重新创建以下图形。这时分面标签的位置如何变化?

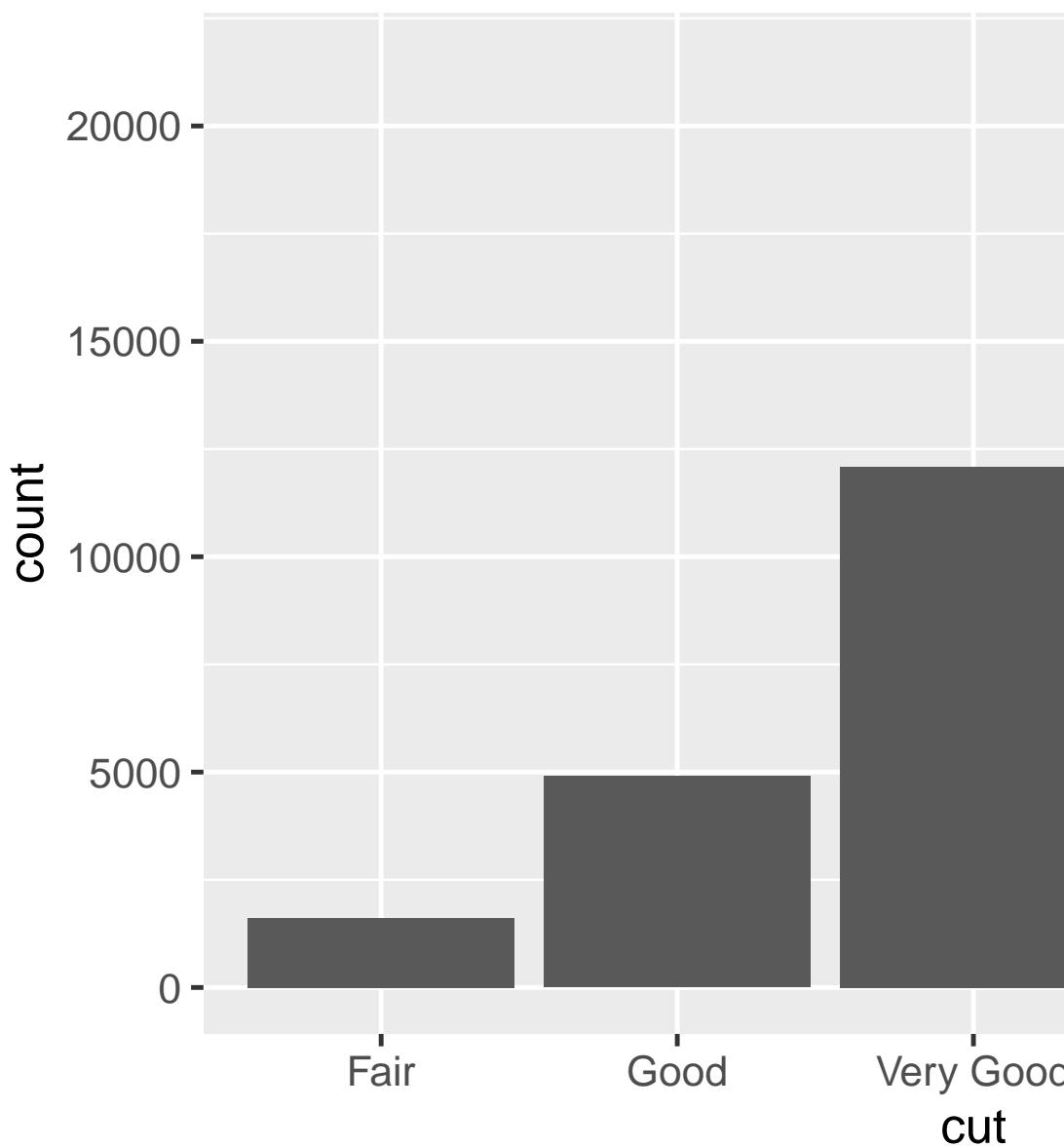
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(drv ~ .)
```

## 9.5 统计变换

下面是一个用 `geom_bar()` 或 `geom_col()` 绘制的基本条形图，图形显示了 `diamonds` 数据集中按 `cut` 分组的钻石总数。`diamonds` 数据集位于 `ggplot2` 包中，包含约 54,000 颗钻石的信息，包括每颗钻石的价格(`price`)、重量(`carat`)、颜色(`color`)、净度(`clarity`) 和切割方式(`cut`)。图形显示，高质量切割的钻石比低质量切割的钻石更多。

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

9 图层



## 9.5 统计变换

在 x 轴上图形显示了来自 `diamonds` 数据集的变量 `cut`。在 y 轴上显示了 `count`, 但 `count` 并不是 `diamonds` 数据集中的变量! `count` 是从哪里来的? 许多图形, 如散点图会绘制数据集的原始值。而其他图形, 如条形图则会计算新的值来绘制:

- 条形图、直方图和频数多边形会将数据进行分箱, 然后绘制每个分箱中的点数 (即分箱计数)。
- 平滑器 (smoothers) 会对您的数据拟合一个模型, 然后绘制该模型的预测值。
- 箱线图会计算分布的五数汇总 (five-number summary), 然后以特殊格式的箱子显示该汇总。

用于计算图形新值的算法被称为 `stat`, 是 statistical transformation (统计变换) 的缩写。@ `fig-vis-stat-bar` 展示了 `geom_bar()` 是如何工作的。

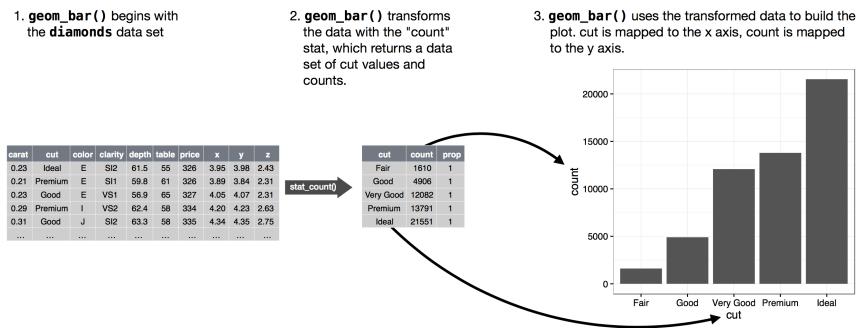


图 9.2: When creating a bar chart we first start with the raw data, then aggregate it to count the number of observations in each bar, and finally map those computed variables to plot aesthetics.

你可以通过检查 `stat` 参数的默认值来了解一个 `geom` 使用的是哪种统计

## 9 图层

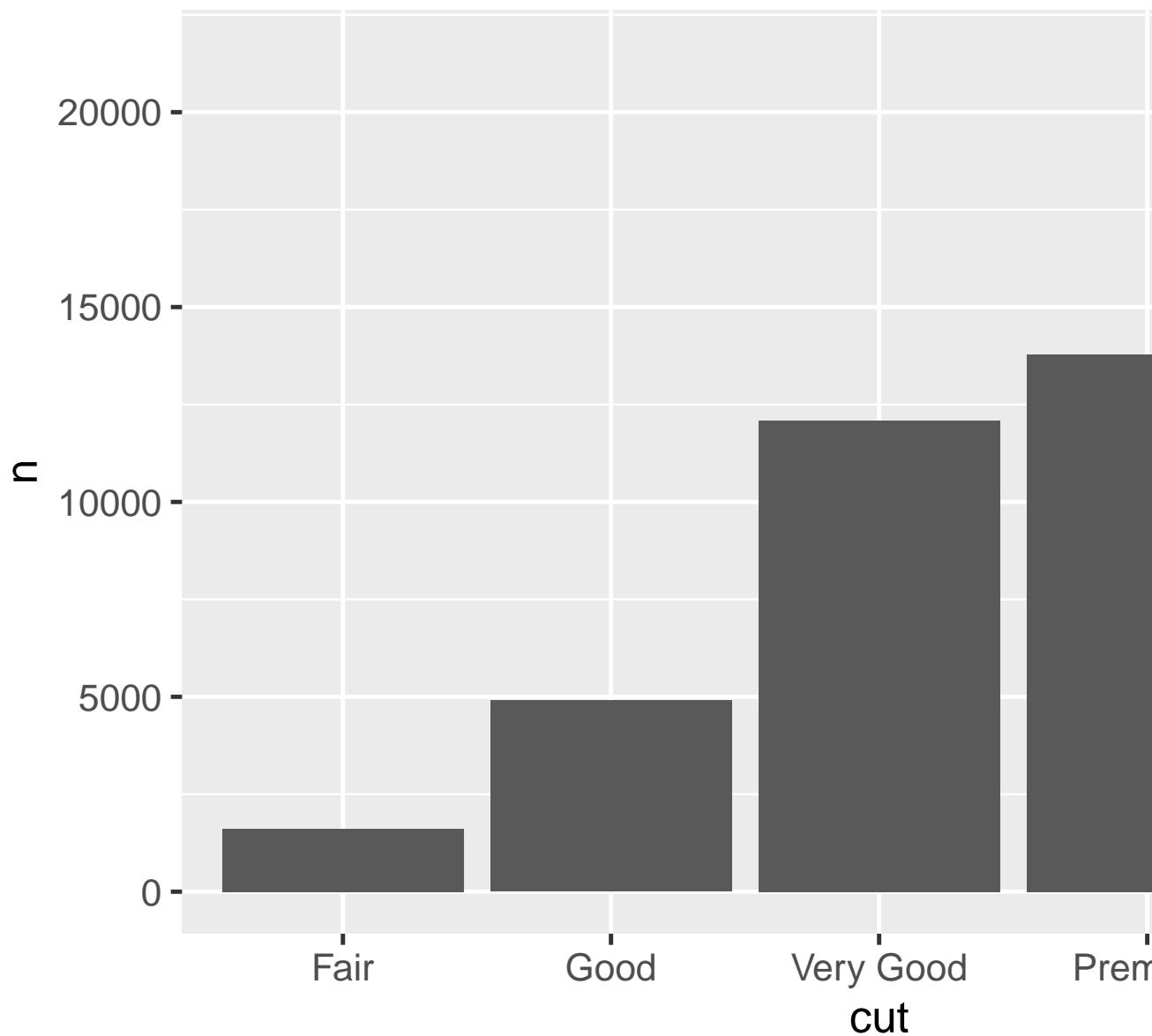
变换。例如?geom\_bar 显示 stat 的默认值是"count"，这意味着 geom\_bar() 使用的是 stat\_count()。stat\_count() 的文档与 geom\_bar() 在同一页面。如果你向下滚动，名为“Computed variables”的部分说明它计算了两个新变量： count 和 prop。

每个 geom 都有一个默认的统计变换；每个统计变换也有一个默认的 geom。这意味着你通常可以使用 geoms 而不用担心底层的统计变换。然而，在三种情况下你可能需要指明使用何种统计变换：

1. 你可能想要覆盖默认的统计变换。在下面的代码中，我们将 geom\_bar() 的统计变换从默认的 count 更改为 identity。这样我们就可以将条形的高度映射到 y 变量的原始值。

```
diamonds |>
  count(cut) |>
  ggplot(aes(x = cut, y = n)) +
  geom_bar(stat = "identity")
```

## 9.5 统计变换

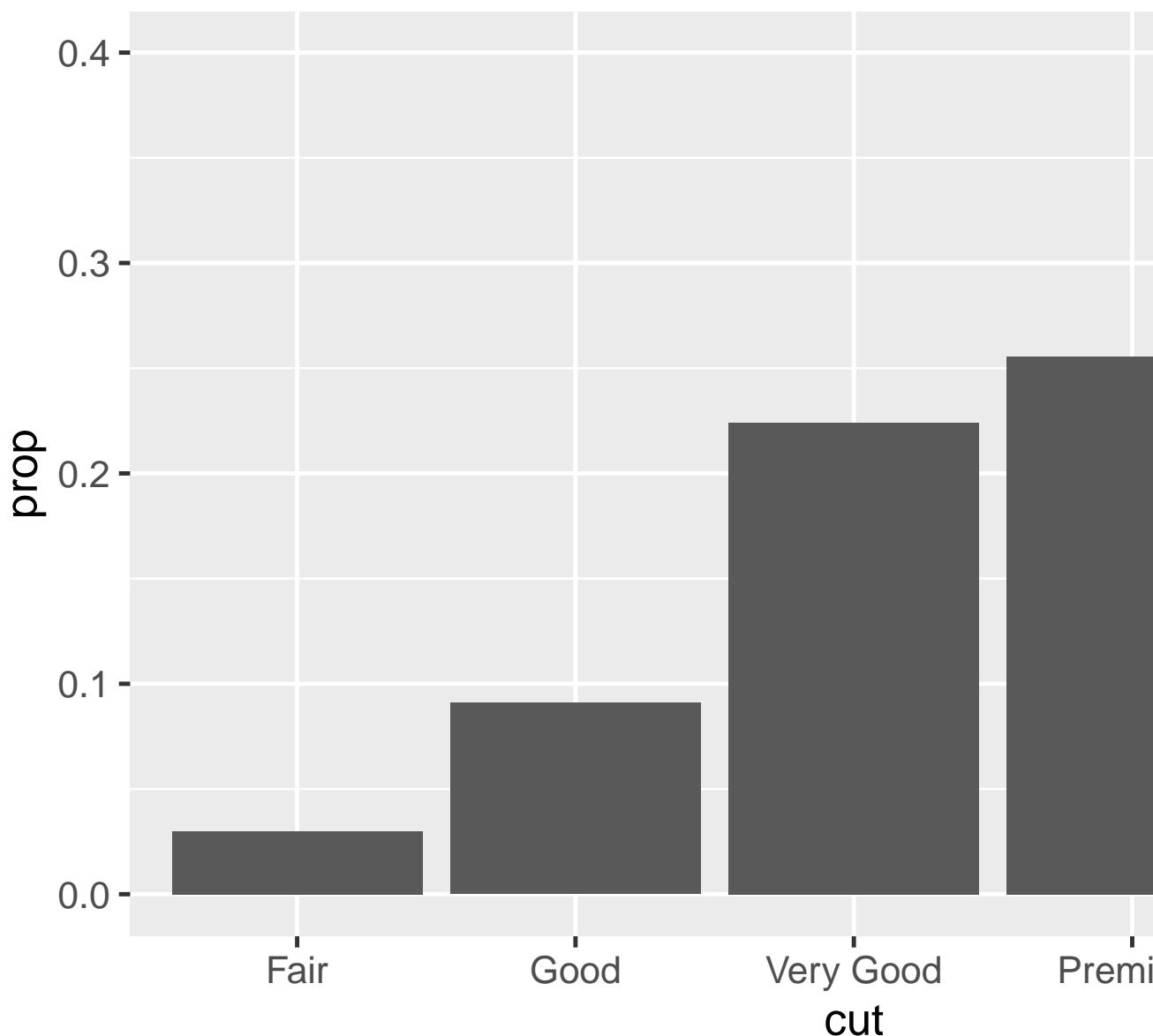


## 9 图层

2. 你可能想要覆盖从变换后的变量到视觉属性的默认映射。例如，你可能想要显示一个比例条形图，而不是计数条形图：

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

## 9.5 统计变换



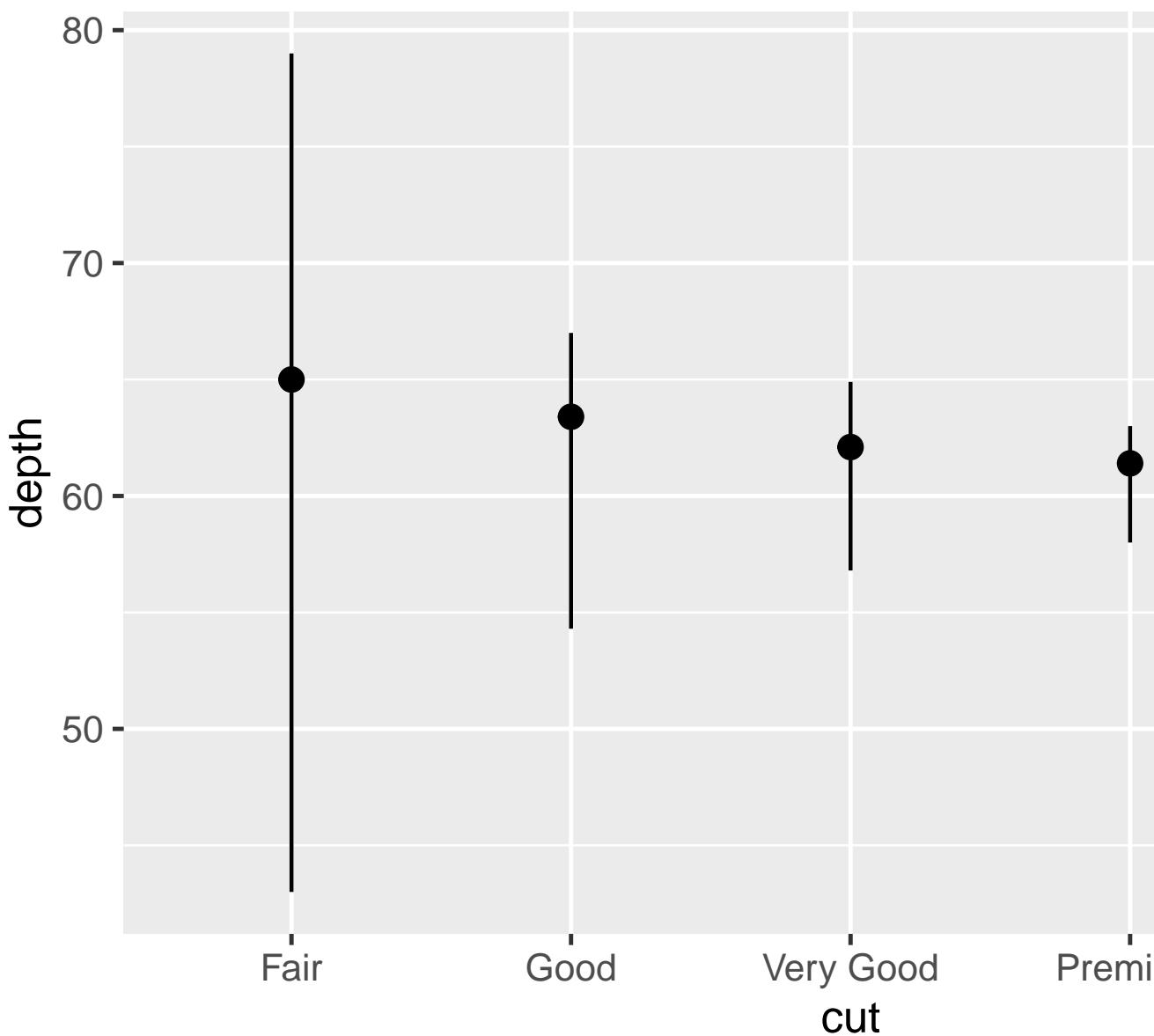
## 9 图层

要找到可以由统计变换计算的可能的变量，请在 `geom_bar()` 的帮助文档中寻找标题为“computed variables”的部分。

3. 你可能希望在你的代码中更加关注统计变换。例如你可能使用 `stat_summary()`，它对每个唯一的 `x` 值汇总 `y` 值，以此来强调你正在计算的汇总统计量：

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

## 9.5 统计变换



## 9 图层

ggplot2 提供了超过 20 种统计变换。每个统计变换都是一个函数，所以你可以通过常规方式获取帮助，例如使用`?stat_bin`。

### 9.5.1 练习

1. `stat_summary()` 的默认 `geom` 是什么？如何使用这个 `geom` 函数而不是 `stat` 函数来重写前面的图形？
2. `geom_col()` 是做什么的？它与 `geom_bar()` 有什么不同？
3. 大多数 geoms 和 stats 都是成对出现的，它们几乎总是协同工作。请列出所有这些对子。它们有什么共同点？（提示：阅读文档。）
4. `stat_smooth()` 函数计算哪些变量？哪些参数控制其行为？
5. 在我们的比例条形图中，我们需要设置 `group = 1`。为什么呢？换句话说，这两个图有什么问题？

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

## 9.6 位置调整

与条形图相关的还有一个神奇的特性。你可以使用颜色美学（color aesthetic）来为条形图上色，但更有用的是，你还可以使用填充美学（fill aesthetic）来为条形图上色：

## 9.6 位置调整

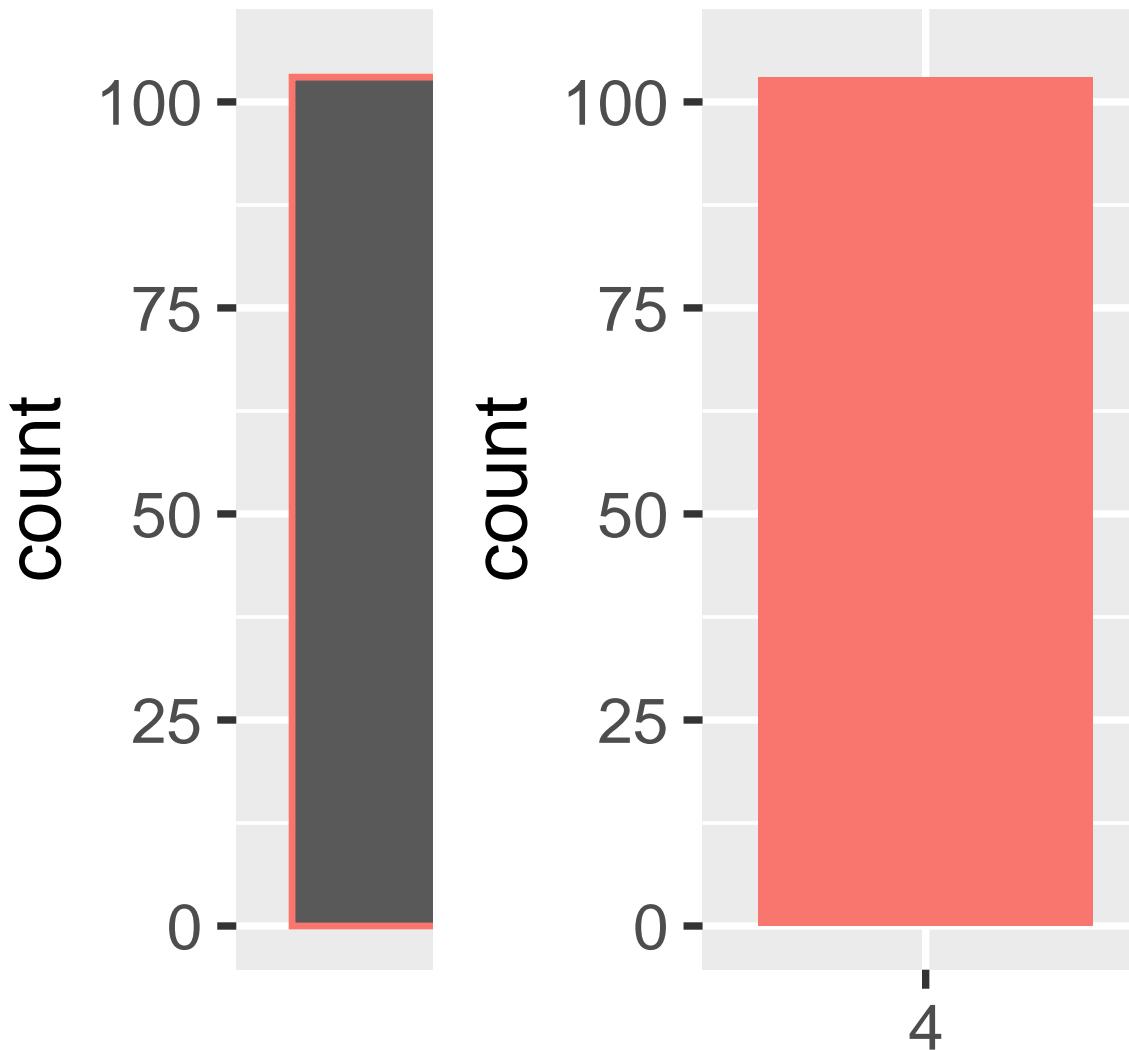
```
# Left
ggplot(mpg, aes(x = drv, color = drv)) +
  geom_bar()

# Right
ggplot(mpg, aes(x = drv, fill = drv)) +
  geom_bar()
```

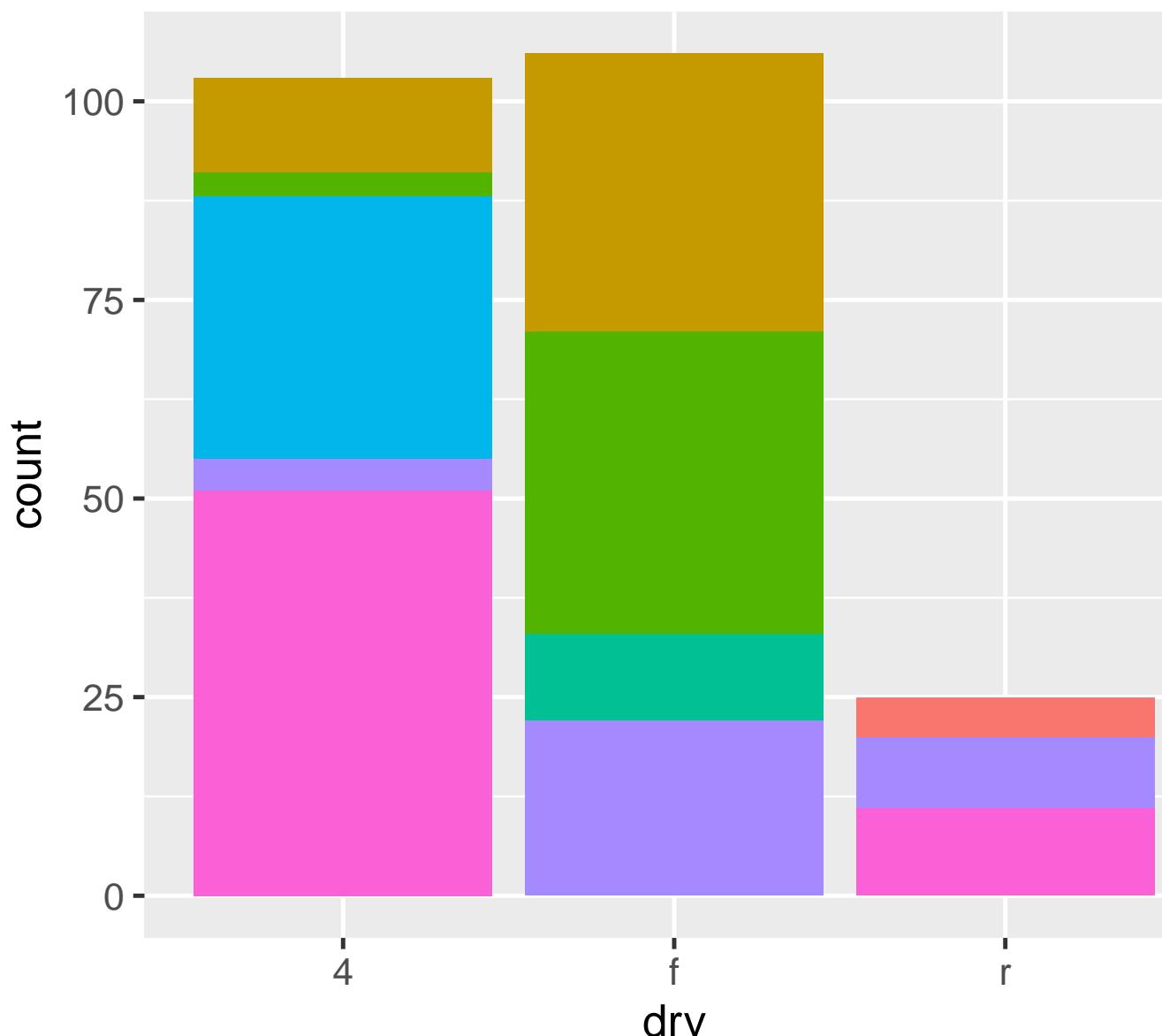
请注意，如果你将填充美学映射到另一个变量，比如 `class`，条形图会自动堆叠。每个彩色矩形代表 `drv` 和 `class` 的一个组合。

```
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar()
```

9 图层



## 9.6 位置调整



269

## 9 图层

堆叠是通过由 `position` 参数指定的位置调整 (position adjustment) 自动执行的。如果你不想创建堆叠条形图, 你可以使用以下三种选项之一: "`identity`"、"`dodge`" 或 "`fill`"。

- `position = "identity"` 会将每个对象精确地放置在其在图中的位置。这对于条形图来说不是很有用, 因为它会使它们重叠。为了看到这种重叠, 我们需要将条形设置为稍微透明, 通过将 `alpha` 设置为一个小值, 或者通过设置 `fill = NA` 使其完全透明。

```
# Left
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(alpha = 1/5, position = "identity")

# Right
ggplot(mpg, aes(x = drv, color = class)) +
  geom_bar(fill = NA, position = "identity")
```

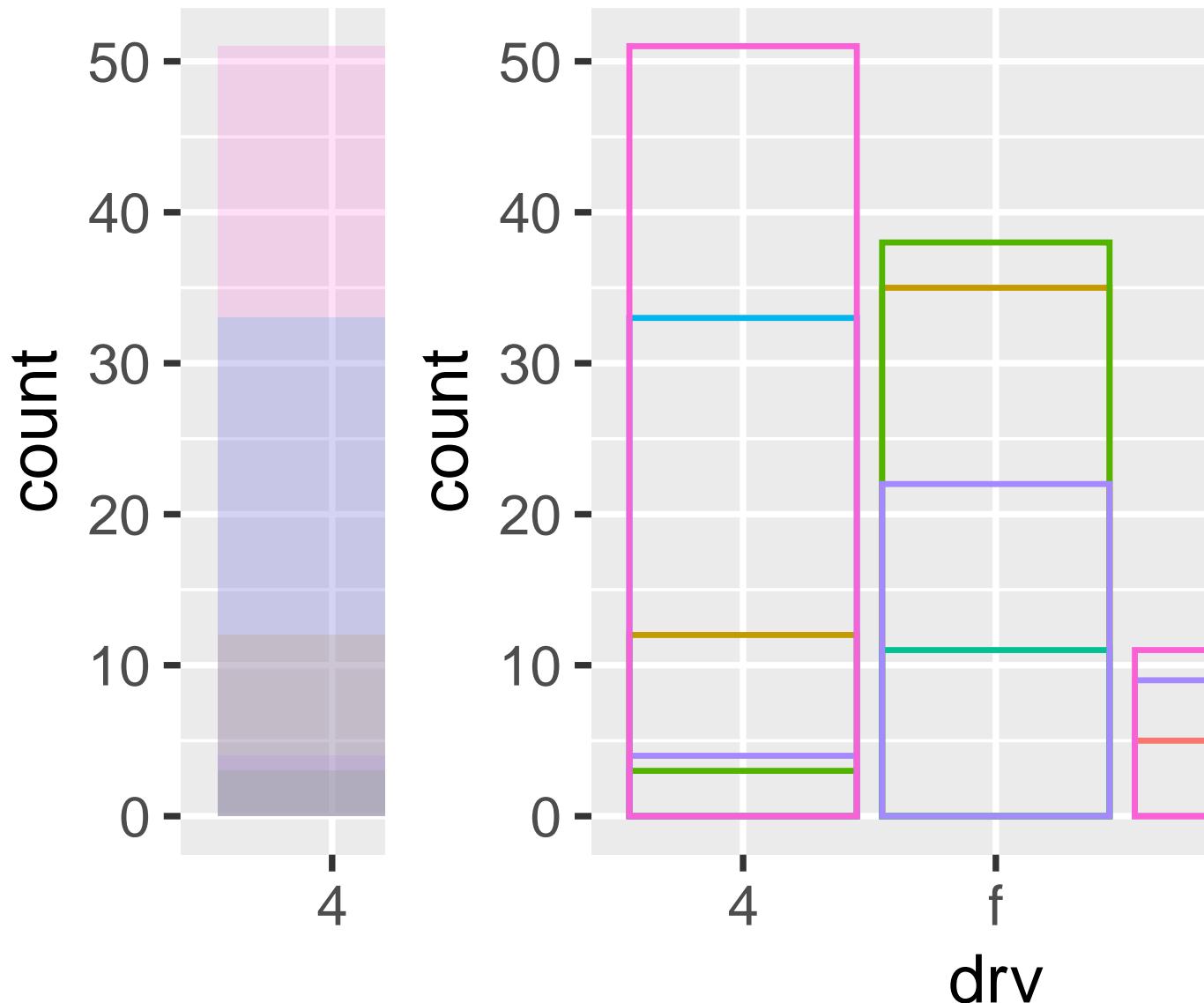
“`identity`” 位置调整对于二维几何对象 (如点) 更为有用, 它是这些对象的默认设置。

- `position = "fill"` 的作用类似于堆叠, 但会使每组堆叠的条形具有相同的高度。这使得比较各组之间的比例更容易。
- `position = "dodge"` 将重叠的对象直接并排放置, 这使得比较各个值更容易。

```
# Left
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(position = "fill")

# Right
```

## 9.6 位置调整

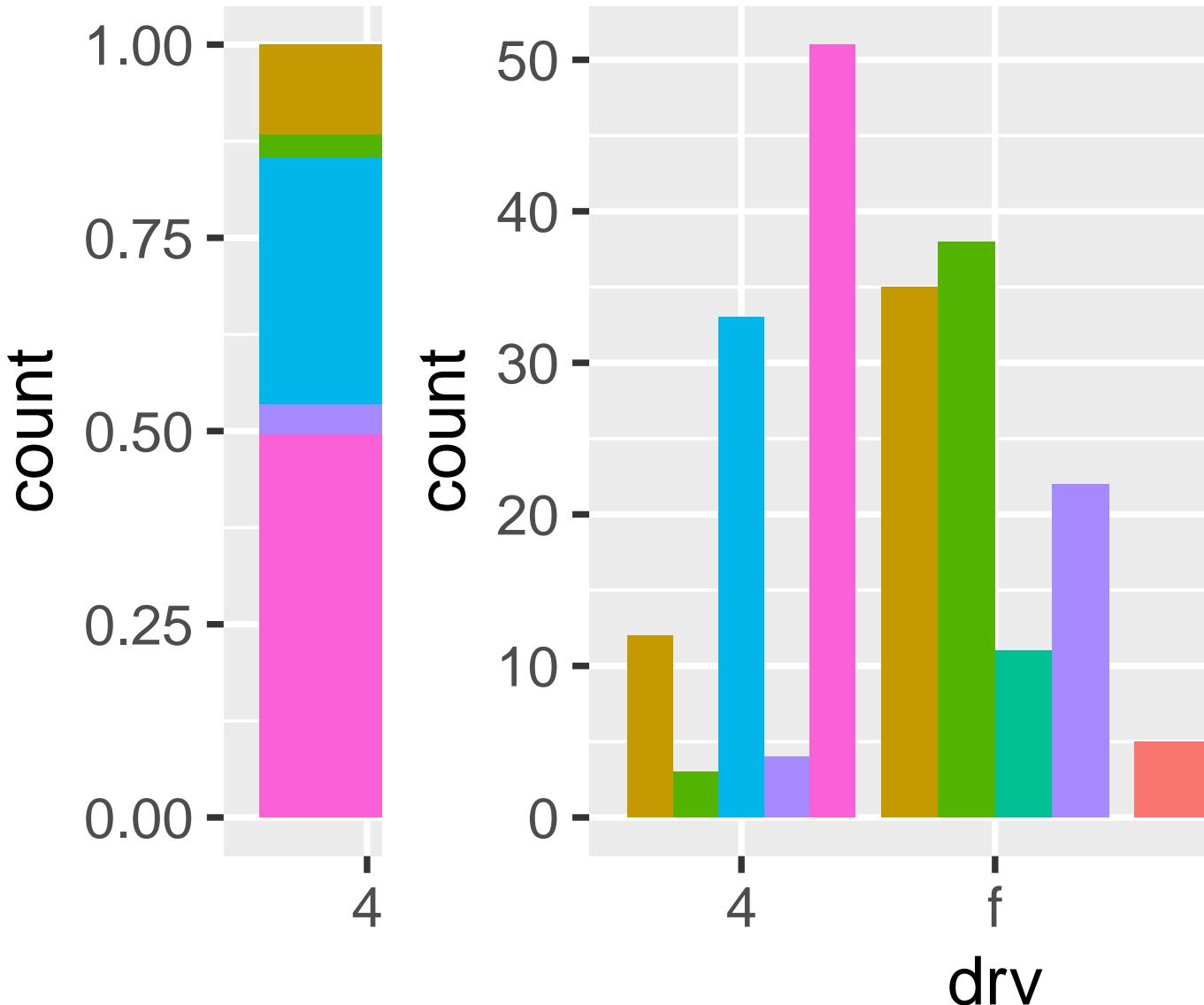


## 9 图层

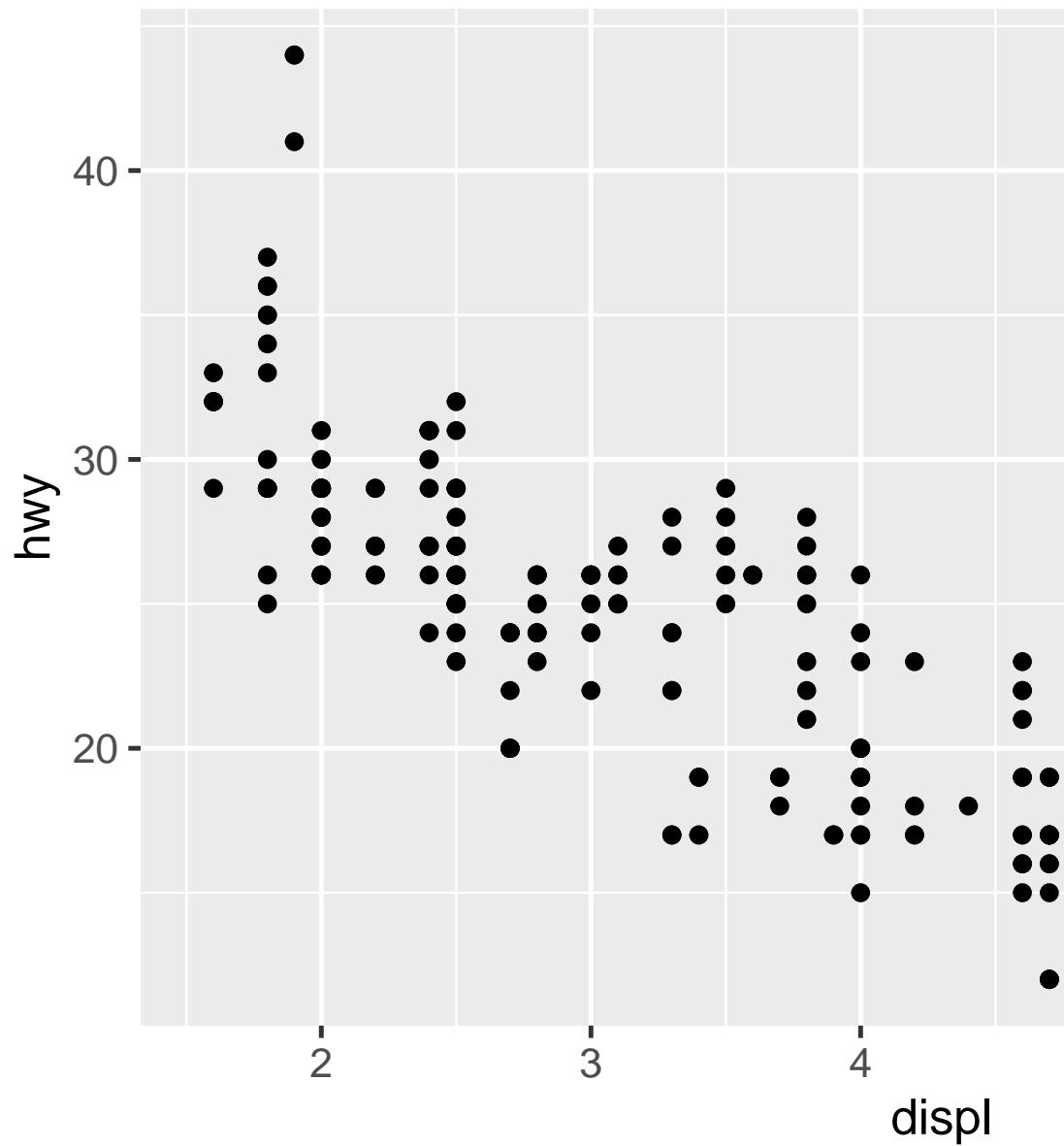
```
ggplot(mpg, aes(x = drv, fill = class)) +  
  geom_bar(position = "dodge")
```

还有一种调整不适用于条形图，但对散点图非常有用。回想我们的第一个散点图。你有没有注意到，尽管数据集中有 234 个观测值，但图中只显示了 126 个点？

## 9.6 位置调整



9 图层



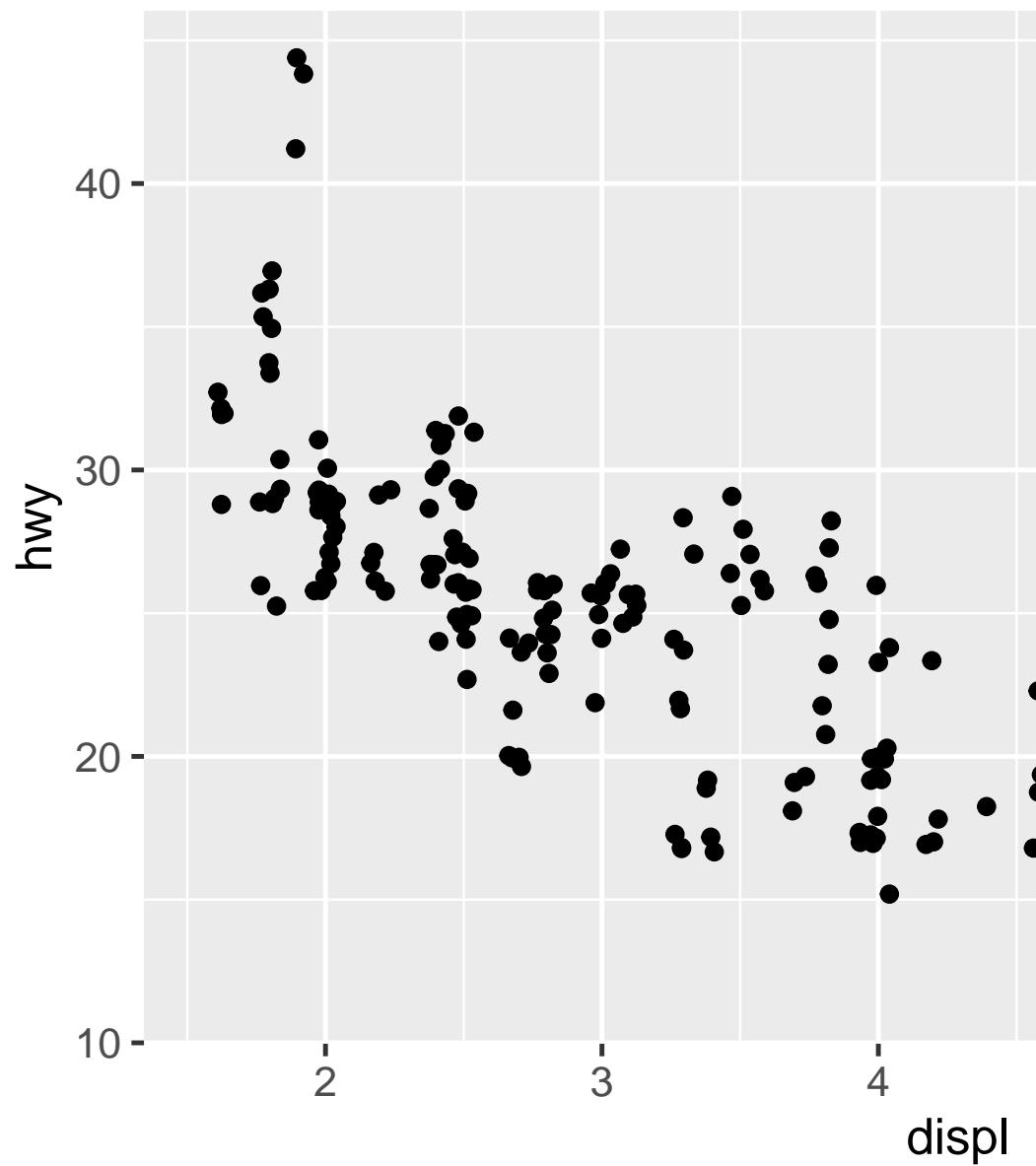
## 9.6 位置调整

`hwy` 和 `displ` 的基础值被四舍五入，因此点显示在一个网格上，许多点互相重叠。这个问题被称为过度绘制（overplotting）。这种布局使得很难看到数据的分布。数据点在整个图中是均匀分布的吗，还是 `hwy` 和 `displ` 的某个特殊组合包含了 109 个值？

你可以通过设置位置调整为“jitter”来避免这种网格化。`position = "jitter"` 会给每个点添加一小部分随机噪声。这会将点分散开来，因为两个点不太可能获得相同数量的随机噪声。

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(position = "jitter")
```

9 图层



## 9.6 位置调整

添加随机性似乎是一种奇怪的改进图表的方式，但尽管它在小范围内降低了图形的准确性，但在大范围内却使图形更清晰。由于这是一种非常有用的操作，`ggplot2` 为 `geom_point(position = "jitter")` 提供了一个简写形式：`geom_jitter()`。

要了解更多关于位置调整的信息，请查阅与每个调整相关的帮助页  
面：`?position_dodge`、`?position_fill`、`?position_identity`、  
`?position_jitter` 和 `?position_stack`。

### 9.6.1 练习

1. 下面的图有什么问题？如何改进它？

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

2. 这两个图之间有什么区别（如果有的话）？为什么？

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point()  
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(position = "identity")
```

3. `geom_jitter()` 中的哪些参数控制抖动量（amount of jittering）？

4. 将 `geom_jitter()` 与 `geom_count()` 进行比较和对比。

5. `geom_boxplot()` 的默认位置调整是什么？使用 `mpg` 数据集创建一个可视化来展示它。

## 9.7 坐标系

坐标系可能是 `ggplot2` 中最复杂的部分。默认的坐标系是笛卡尔坐标系，其中 `x` 和 `y` 位置独立地确定每个点的位置。还有另外两种坐标系在某些情况下也很有用。

- `coord_quickmap()` 为地理地图设置了正确的纵横比。如果你使用 `ggplot2` 绘制空间数据，这一点非常重要。本书中没有空间讨论地图，但你可以在《*ggplot2: Elegant graphics for data analysis*》一书的[Maps chapter](#) 中了解更多信息。

```
nz <- map_data("nz")

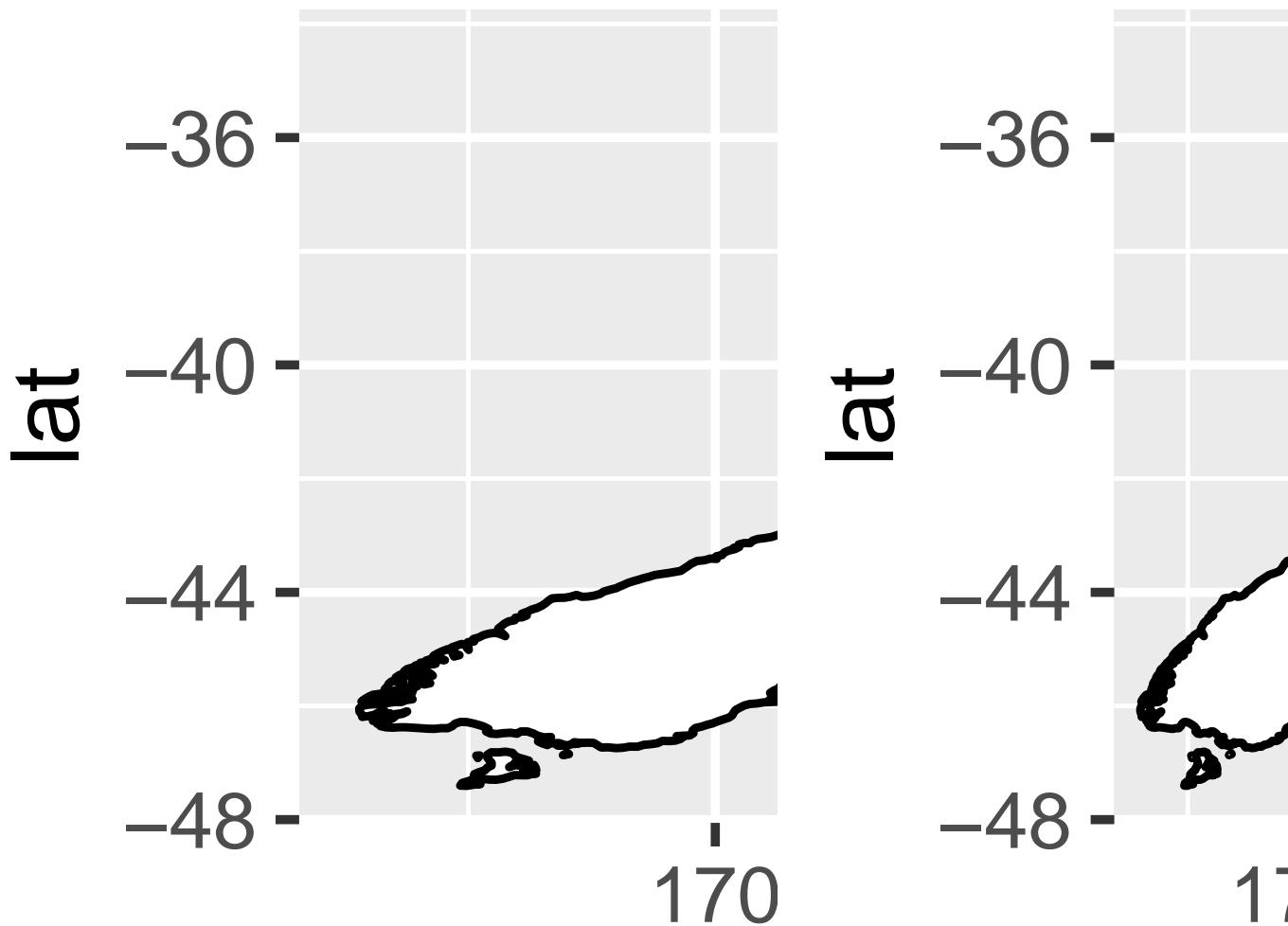
ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black")

ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black") +
  coord_quickmap()
```

- `coord_polar()` 使用极坐标系。极坐标系揭示了条形图和扇形图（Coxcomb chart）之间一个有趣的联系。

```
bar <- ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = clarity, fill = clarity),
    show.legend = FALSE,
    width = 1
  ) +
```

9.7 坐标系



## 9 图层

```
theme(aspect.ratio = 1)

bar + coord_flip()
bar + coord_polar()
```

### 9.7.1 练习

1. 使用 `coord_polar()` 将堆叠的条形图转换为饼状图。
2. `coord_quickmap()` 和 `coord_map()` 之间的区别是什么？
3. 下图告诉你 `city` 和高速公路 `mpg` 之间的关系是什么？为什么 `coord_fixed()` 很重要？`geom_abline()` 的作用什么？

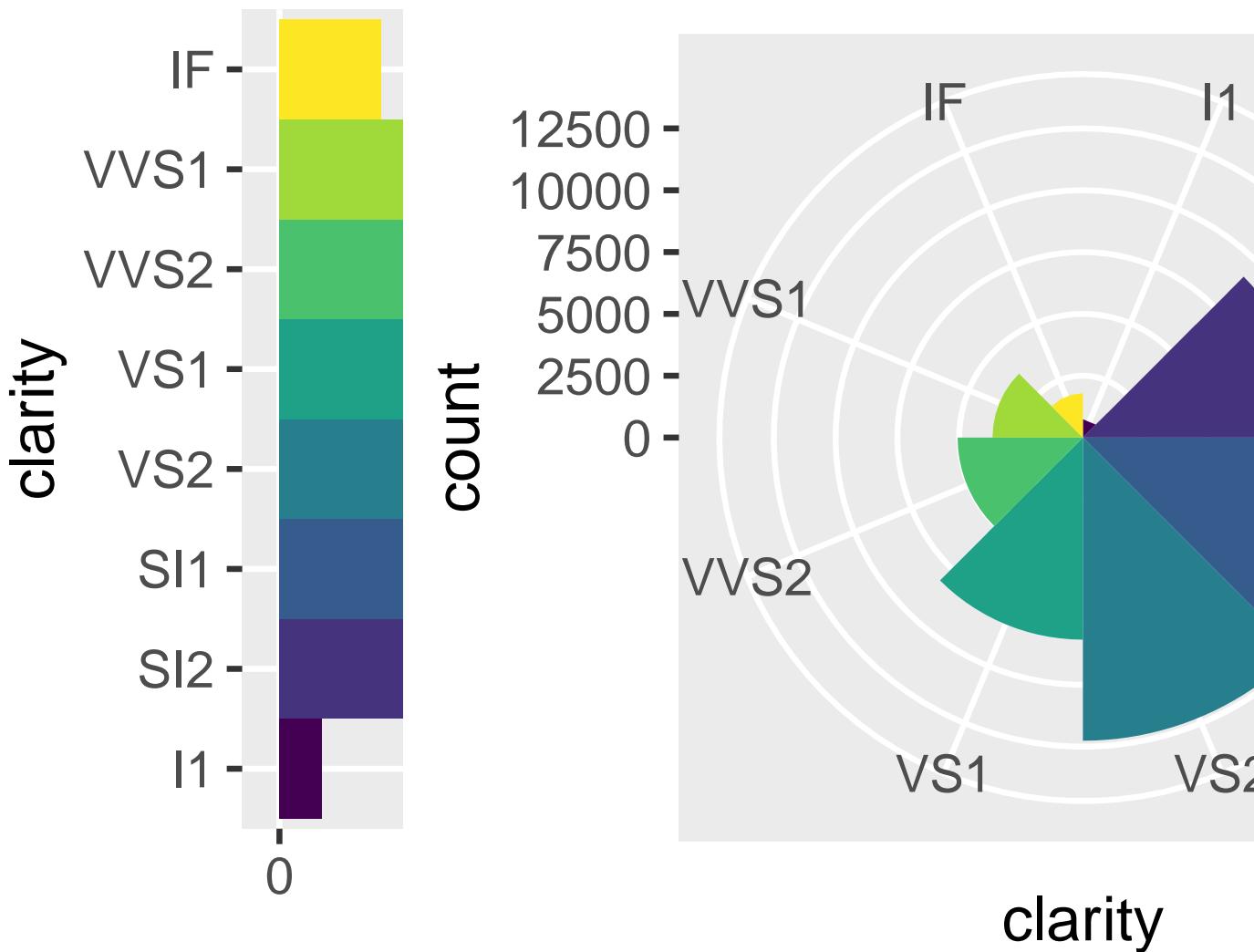
```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_fixed()
```

## 9.8 图形的分层语法

我们可以通过添加位置调整、统计量、坐标系和分面来扩展你在 @sec-ggplot2-calls 学到的绘图模板：

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

## 9.8 图形的分层语法



## 9 图层

```
stat = <STAT>,
position = <POSITION>
) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION>
```

我们的新模板需要七个参数，即模板中出现的括号中的词。在实践中，你很少需要为制作图形提供所有七个参数，因为 `ggplot2` 会为除了数据、映射和 `geom` 函数之外的所有内容提供有用默认值。

模板中的七个参数构成了图形的语法 (grammar of graphics)，这是一个用于构建图形的正式系统。图形的语法基于这样一个见解：你可以将任何图形唯一地描述为数据集、`geom`、一组映射、统计量、位置调整、坐标系、分面方案和主题的组合。

为了了解这是如何工作的，请考虑如何从零开始构建一个基本的图形：你可以从一个数据集开始，然后将其转换为你想要显示的信息（使用统计量）。接下来，你可以选择一个几何对象来表示转换后数据中的每个观测值。然后，你可以使用 `geom` 的美学属性来表示数据中的变量。你可以将每个变量的值映射到美学的水平。这些步骤在图 ?? 中进行了说明。然后，你会选择一个坐标系来放置 `geom`，使用对象的位置（它本身也是一个美学属性）来显示 `x` 和 `y` 变量的值。

在这一点上，你将拥有一个完整的图形，但你可以进一步调整 `geom` 在坐标系内的位置（位置调整）或将图形拆分为子图（分面）。你还可以通过添加一个或多个附加层来扩展图形，其中每个附加层都使用数据集、`geom`、一组映射、统计量和位置调整。

你可以使用这种方法来构建你想象中的任何图形。换句话说，你可以使用本章中学习的代码模板来构建数十万个独特的图形。

## 9.9 小结

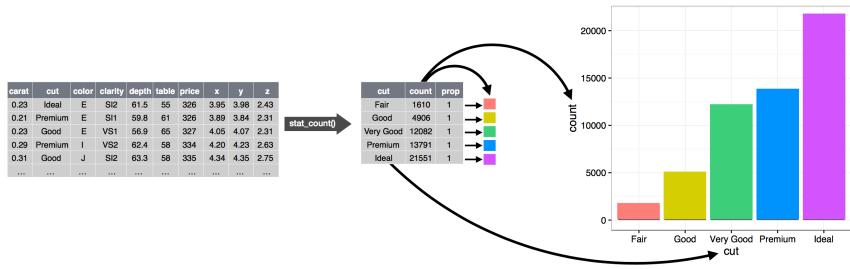


图 9.3: Steps for going from raw data to a table of frequencies to a bar plot where the heights of the bar represent the frequencies。

如果你想了解更多关于 ggplot2 的理论基础, 你可能会喜欢阅读《[The Layered Grammar of Graphics](#)》这篇科学论文, 它详细描述了 ggplot2 的理论。

## 9.9 小结

在本章中, 你从美学和几何图形开始学习了图形的分层语法, 以构建简单的图形、通过分面将图形拆分为子集、通过统计量了解 geom 是如何计算的、通过位置调整控制 geom 可能重叠时的位置细节, 以及通过坐标系统从根本上改变 x 和 y 的含义。我们尚未涉及的一个层次是主题 (theme), 我们将在小节 ?? 中介绍。

两个非常有用的资源可以帮助你全面了解 ggplot2 的完整功能, 分别是 ggplot2 速查表(你可以在<https://posit.co/resources/cheatsheets>找到)和 ggplot2 包网站 (<https://ggplot2.tidyverse.org/>)。

你应该从本章中学到的一个重要教训是, 当你觉得需要 ggplot2 没有提供的 geom 时, 最好先查看是否有人已经通过创建提供该 geom 的 ggplot2 扩展包来解决了你的问题。



# 10 探索性数据分析

## 10.1 引言

本章将向你展示如何系统地使用可视化和转换来探索数据，这项任务被统计学家称为探索性数据分析（Exploratory Data Analysis，简称 EDA）。EDA 是一个迭代循环的过程。你需要：

1. 对你的数据产生疑问。
2. 通过数据可视化、转换和建模来寻找答案。
3. 利用所学的知识来完善你的问题或者产生新的问题。

EDA 并不是一个有着严格规则的正式流程，其更多的是一种思维方式。在 EDA 的初始阶段，你应该自由地调查任何出现在你脑海中的想法。这些想法中的一些会成功，而一些则会是死胡同。随着你的探索继续，你将专注于一些特别有效的见解，并最终将其记录下来并与他人交流。

EDA 是任何数据分析的重要组成部分，即使主要的研究问题已经被直接提出，因为你总是需要调查你的数据质量。数据清洗只是 EDA 的一个应用：你问的问题是数据是否符合你的期望。为了进行数据清洗，你需要运用 EDA 的所有工具：可视化、转换和建模。

### 10.1.1 必要条件

在本章中，我们将结合你学到的关于 dplyr 和 ggplot2 的知识，以交互的方式提出问题，用数据来回答问题，然后提出新的问题。

```
library(tidyverse)
```

## 10.2 问题

“没有常规的统计问题，只有可疑的统计常规。” — Sir David Cox

“一个对正确问题的近似答案（这个问题常常是模糊的）远胜于一个对错误问题的精确答案（这个问题总是可以精确化的）。” — John Tukey

在 EDA 期间，你的目标是了解你的数据。要做到这一点，最简单的方法是将问题作为工具来指导你的调查。当你提出一个问题时，这个问题会将你的注意力集中在数据集的特定部分上，并帮助你决定制作哪些图形、模型或转换。

EDA 本质上是一个创造性过程。和大多数创造性过程一样，提出高质量问题的关键是生成大量问题。在你的分析开始时，很难提出有启示性的问题，因为你不知道可以从数据集中获取哪些见解。另一方面，你提出的每一个新问题都会让你了解数据的一个新方面，并增加你发现新事物的机会。如果你根据所发现的内容，针对每个问题都提出一个新问题，你就能迅速深入数据中最有趣的部分，并开发出一系列引人深思的问题。

没有规定你应该提出哪些问题来指导你的研究。然而，两种类型的问题始终有助于在你的数据中发现新事物。你可以大致地将这些问题表述为：

1. 我的变量中存在什么类型的变异?
2. 我的变量之间存在什么类型的协变?

本章的其余部分将探讨这两个问题。我们将解释什么是变异和协变，并向你展示几种回答问题的方法。

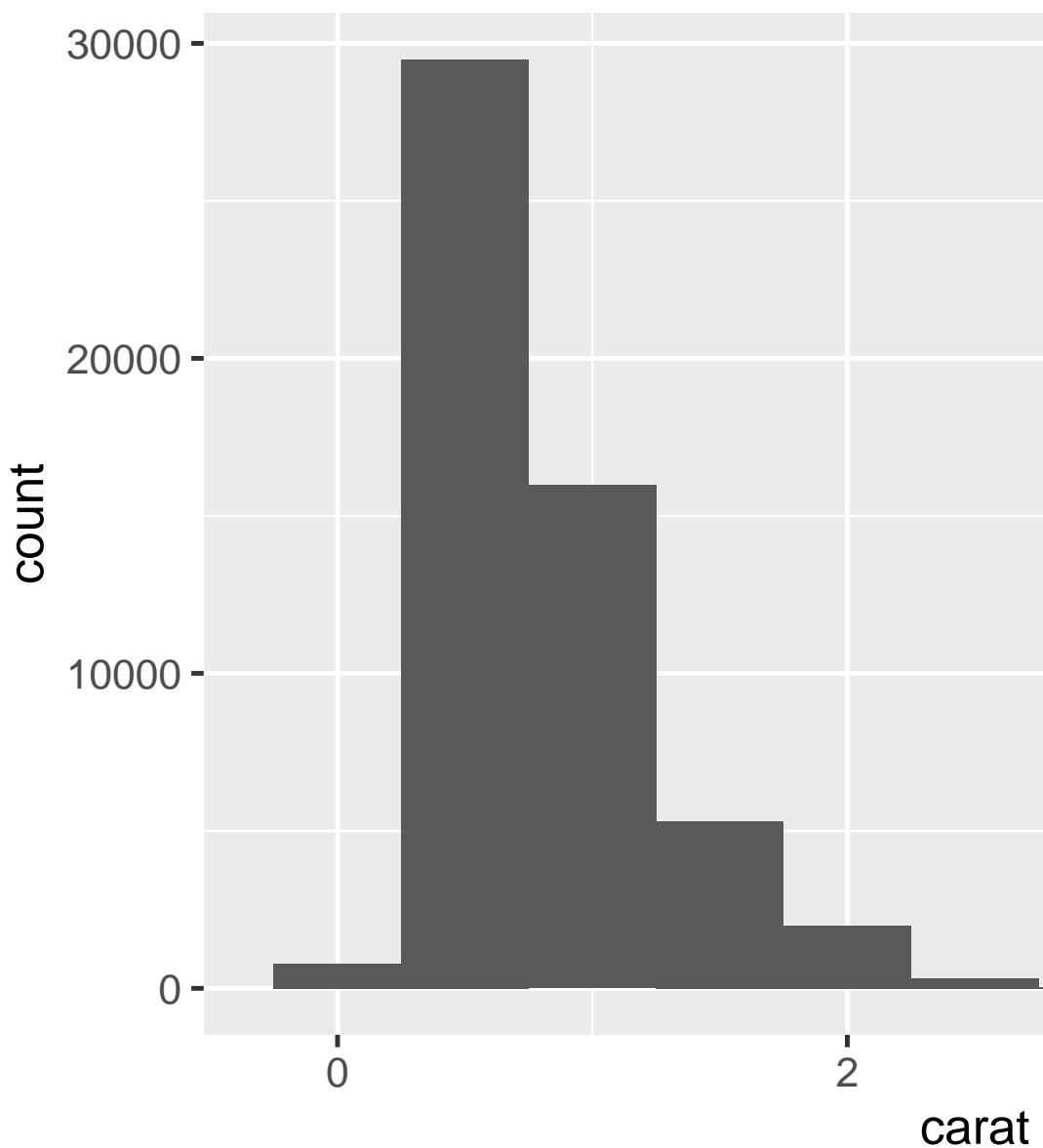
## 10.3 变异

变异 (variation) 是指变量的值从一次测量到另一次测量时发生变化的趋势。在现实生活中，你很容易看到变异；如果你对任何连续变量进行两次测量，你会得到两个不同的结果。即使你测量的是像光速这样的常量，也是如此。你的每次测量都会包含一定量的误差，这些误差会随着测量的不同而有所变化。变量也可能因为测量对象的不同（例如，不同人的眼睛颜色）或时间的不同（例如，电子在不同时刻的能量水平）而发生变化。每个变量都有自己特有的变异模式，这些模式可以揭示出关于该变量在同一观测值的不同测量之间以及不同观测值之间的有趣信息。理解这种模式的最好方法是可视化变量的值分布，这是你在章节 ?? 中已经学过的内容。

我们将通过可视化 `diamonds` 数据集中约 54,000 颗钻石的重量 (`carat`) 分布来开始我们的探索。由于 `carat` 是一个数值变量，我们可以使用直方图来表示。

```
ggplot(diamonds, aes(x = carat)) +  
  geom_histogram(binwidth = 0.5)
```

10 探索性数据分析



既然你现在可以可视化变异了，你应该在你的图中寻找什么？你应该提出什么类型的后续问题？我们已经在下方列出了你在图形中最有用的信息类型，并为每种类型的信息提供了一些后续问题。提出好的后续问题的关键在于依靠你的好奇心（你想更多了解什么？）和怀疑精神（这可能会误导人吗？）。

### 10.3.1 典型值

在条形图和直方图中，高的条形代表变量的常见值，较短的条形代表不太常见的值。没有条形的地方表示在您的数据中未看到的值。为了将这些信息转化为有用的问题，请寻找任何出乎意料的东西：

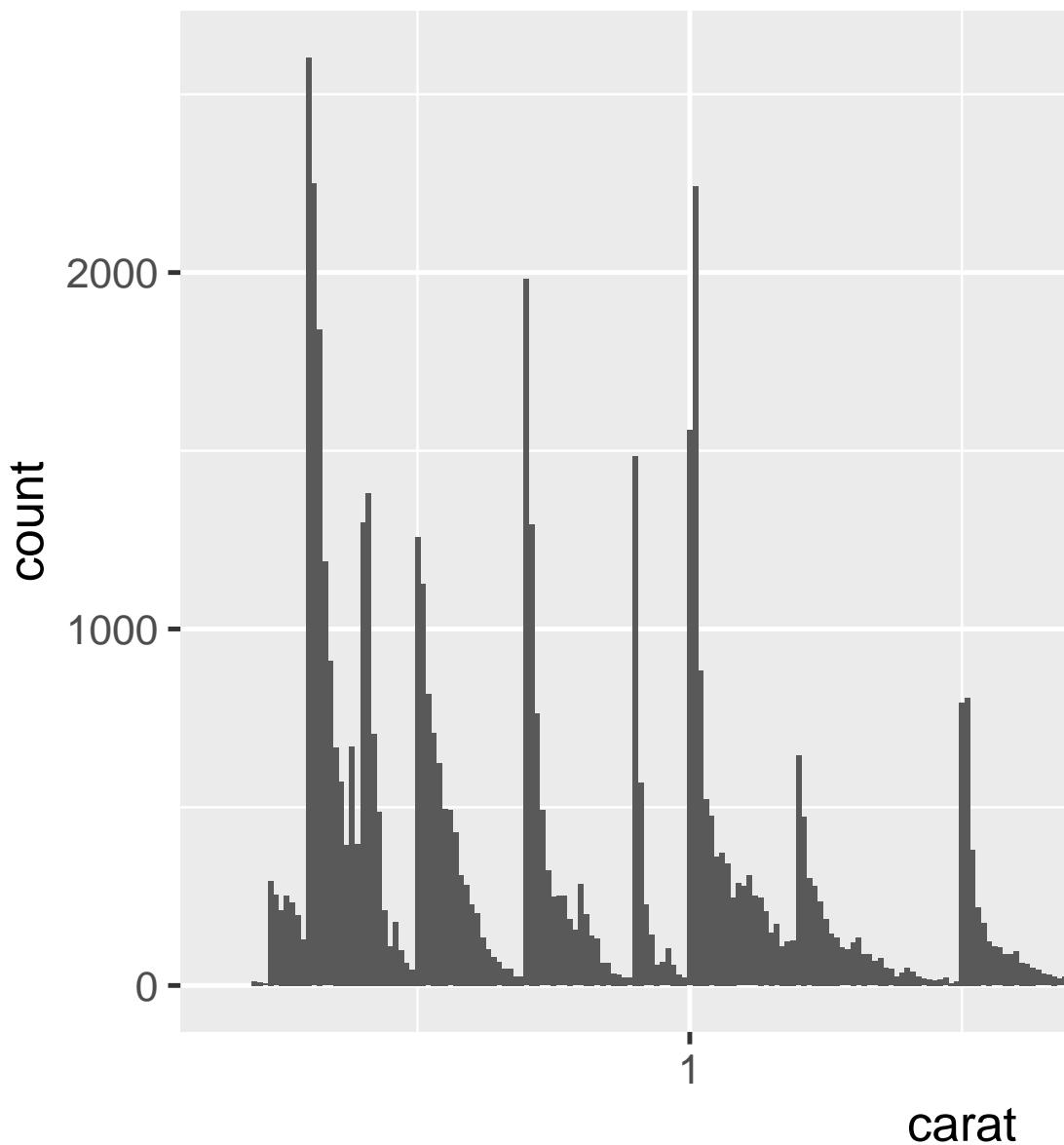
- 哪些值是最常见的？为什么？
- 哪些值是罕见的？为什么？这符合你的预期吗？
- 你是否看到任何不寻常的模式？可能是什么原因导致的？

让我们看一下较小钻石的 `carat` 分布。

```
smaller <- diamonds |>
  filter(carat < 3)

ggplot(smaller, aes(x = carat)) +
  geom_histogram(binwidth = 0.01)
```

10 探索性数据分析



这个直方图提出了几个有趣的问题：

- 为什么整克拉和常见的小数克拉的钻石更多？
- 为什么每个峰值右侧的钻石比左侧的稍微多一些？

可视化图表还可以揭示集群，这表明你的数据中存在子组。为了理解这些子组，请问：

- 每个子组内的观测值如何相互相似？
- 不同集群中的观测值如何相互不同？
- 你如何解释或描述这些集群？
- 为什么集群的出现可能会产生误导？

其中一些问题可以用数据来回答，而一些问题则需要关于数据的领域专业知识。许多问题将促使你探索变量之间的关系，例如，查看一个变量的值是否可以解释另一个变量的行为。我们稍后会讲到这一点。

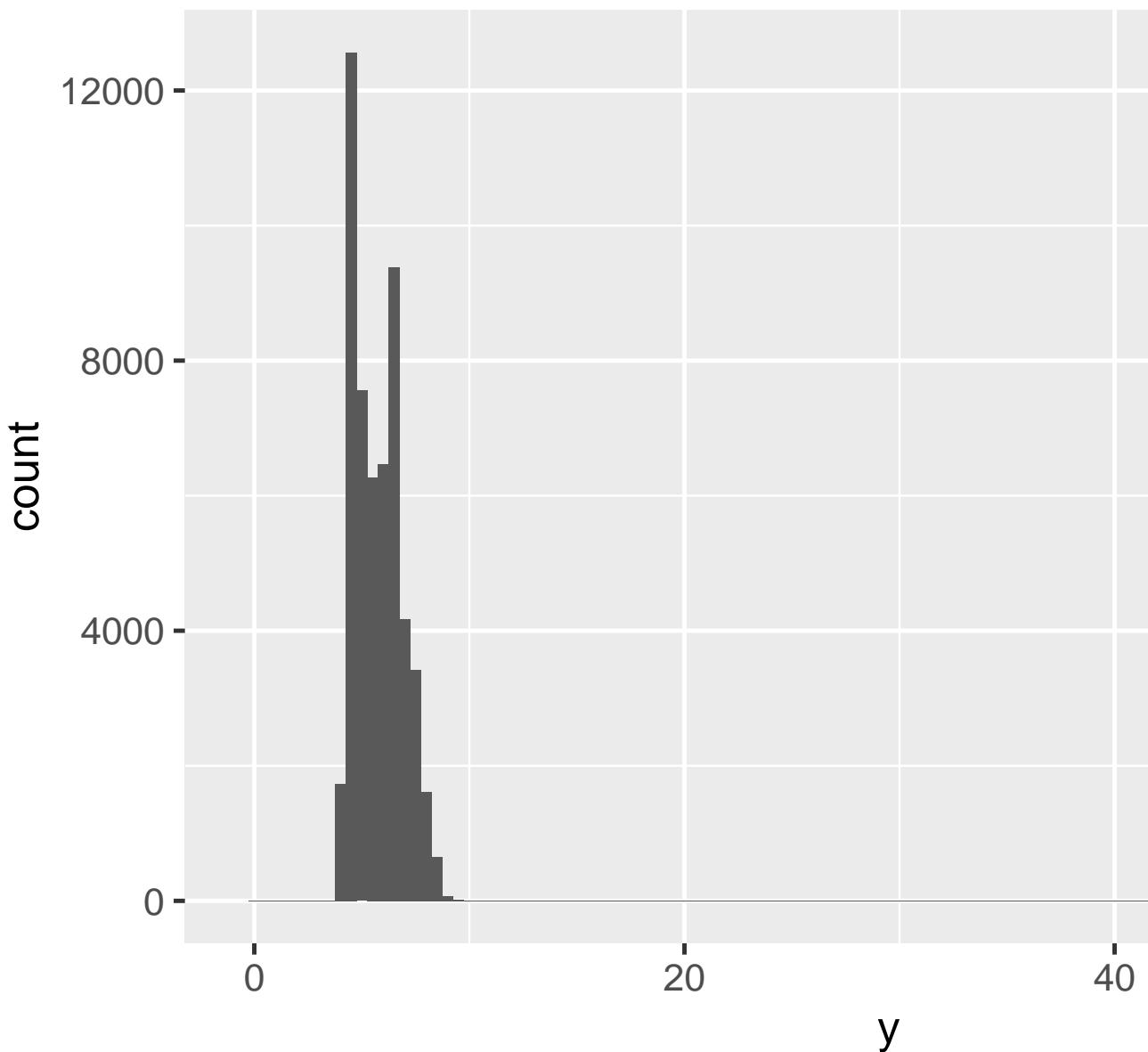
### 10.3.2 异常值

异常值是不寻常的观测值；它们是似乎不符合模式的数据点。有时异常值是数据输入错误，有时它们只是在这组数据收集中偶然观察到的极端值，而其他时候它们可能暗示着重要的新发现。当你拥有大量数据时，有时在直方图中很难看到异常值。例如，查看钻石数据集中  $y$  变量的分布。异常值存在的唯一证据是  $x$  轴上异常宽的界限。

## 10 探索性数据分析

```
ggplot(diamonds, aes(x = y)) +  
  geom_histogram(binwidth = 0.5)
```

10.3 变异



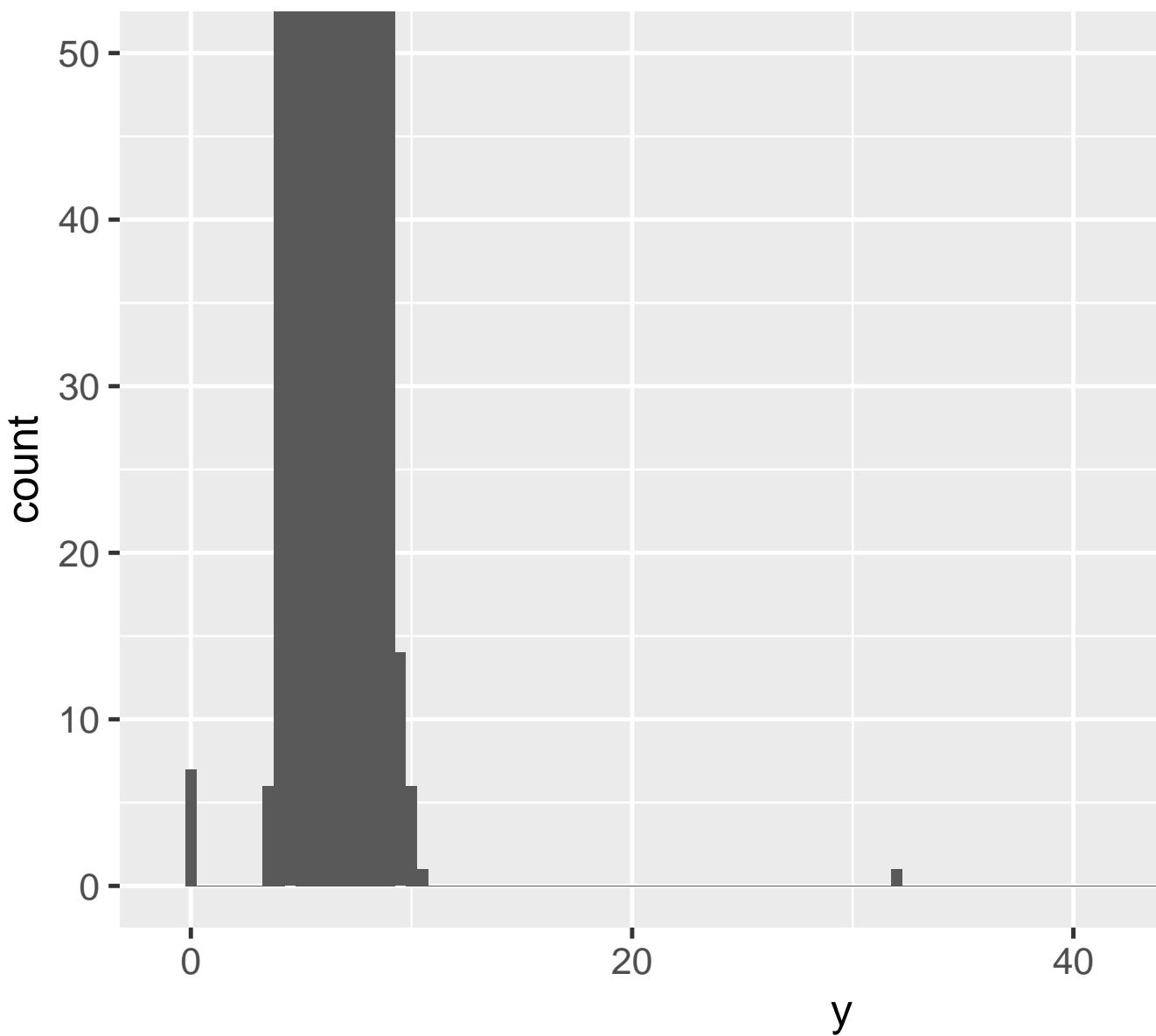
293

## 10 探索性数据分析

在常见的数值范围（箱子）中有许多观测值，这使得罕见的数值范围（箱子）非常短，因此很难看到它们（尽管也许如果你紧盯着 y 轴的 0 点仔细看，会发现点什么）。为了更容易地看到异常值，我们需要使用 `coord_cartesian()` 函数将 y 轴缩放到较小的值域范围。

```
ggplot(diamonds, aes(x = y)) +  
  geom_histogram(binwidth = 0.5) +  
  coord_cartesian(ylim = c(0, 50))
```

10.3 变异



295

## 10 探索性数据分析

`coord_cartesian()` 还有一个 `xlim()` 参数，当你需要放大 x 轴时可以使用它。`g gplot2` 同样有 `xlim()` 和 `ylim()` 函数，但它们的工作方式略有不同：它们会丢弃超出限制范围的数据。

这使得我们能够看到有三个异常值：0, ~30 和 ~60。我们使用 `dplyr` 将它们筛选出来。

```
unusual <- diamonds |>
  filter(y < 3 | y > 20) |>
  select(price, x, y, z) |>
  arrange(y)

unusual
#> # A tibble: 9 x 4
#>   price     x     y     z
#>   <int> <dbl> <dbl> <dbl>
#> 1 5139     0     0     0
#> 2 6381     0     0     0
#> 3 12800    0     0     0
#> 4 15686    0     0     0
#> 5 18034    0     0     0
#> 6 2130     0     0     0
#> 7 2130     0     0     0
#> 8 2075    5.15  31.8  5.12
#> 9 12210   8.09  58.9  8.06
```

`y` 变量测量的是这些钻石的三个维度之一，单位为毫米。我们知道钻石的宽度不可能为 0 毫米，所以这些值一定是错误的。通过进行探索性数据分析 (EDA)，我们发现了被编码为 0 的缺失数据，而我们仅仅通过搜索 `NAs` 是无法发现这些数据的。接下来，我们可能会选择将这些值重新编码为 `NAs`，以防止误

导性的计算。我们也可能怀疑 32 毫米和 59 毫米的测量值是不合理的：这些钻石的长度超过了一英寸，但价格并没有达到数万美元！

一个好的做法是在包含和不包含异常值的情况下重复你的分析。如果它们对结果的影响很小，而且你无法确定它们出现的原因，那么忽略它们并继续分析是合理的。然而，如果它们对你的结果有重大影响，你不应该没有理由就删除它们。你需要找出导致它们出现的原因（例如数据输入错误），并在你的报告中披露你已经删除了这些值。

### 10.3.3 练习

- 探索 `diamonds` 数据集中变量 `x`、`y` 和 `z` 的分布。你发现了什么？想象一下钻石，并思考你如何决定哪个维度是长度、宽度和深度。
- 探索 `price` 的分布。你是否发现了任何不寻常或令人惊讶的事情？（提示：仔细考虑 `binwidth`，并确保你尝试了一系列不同的值。）
- 0.99 克拉的钻石有多少颗？1 克拉的钻石有多少颗？你认为这种差异的原因是什么？
- 在直方图上放大时，比较 `coord_cartesian()` 和 `xlim()` 或 `ylim()` 的区别。如果你没有设置 `binwidth` 会发生什么？如果你尝试缩放到只显示半根柱子时会发生什么？

## 10.4 异常值

如果你在数据集中遇到了异常值，并且只是想继续剩余分析，那么你有两个选择。

## 10 探索性数据分析

1. 删除包含异常值的整行:

```
diamonds2 <- diamonds |>  
filter(between(y, 3, 20))
```

我们不推荐这个选项，因为一个无效的值并不意味着该观测值的其他所有值也都是无效的。此外，如果你的数据质量较低，当你将这种方法应用于每个变量时，你可能会发现你没有留下任何数据！

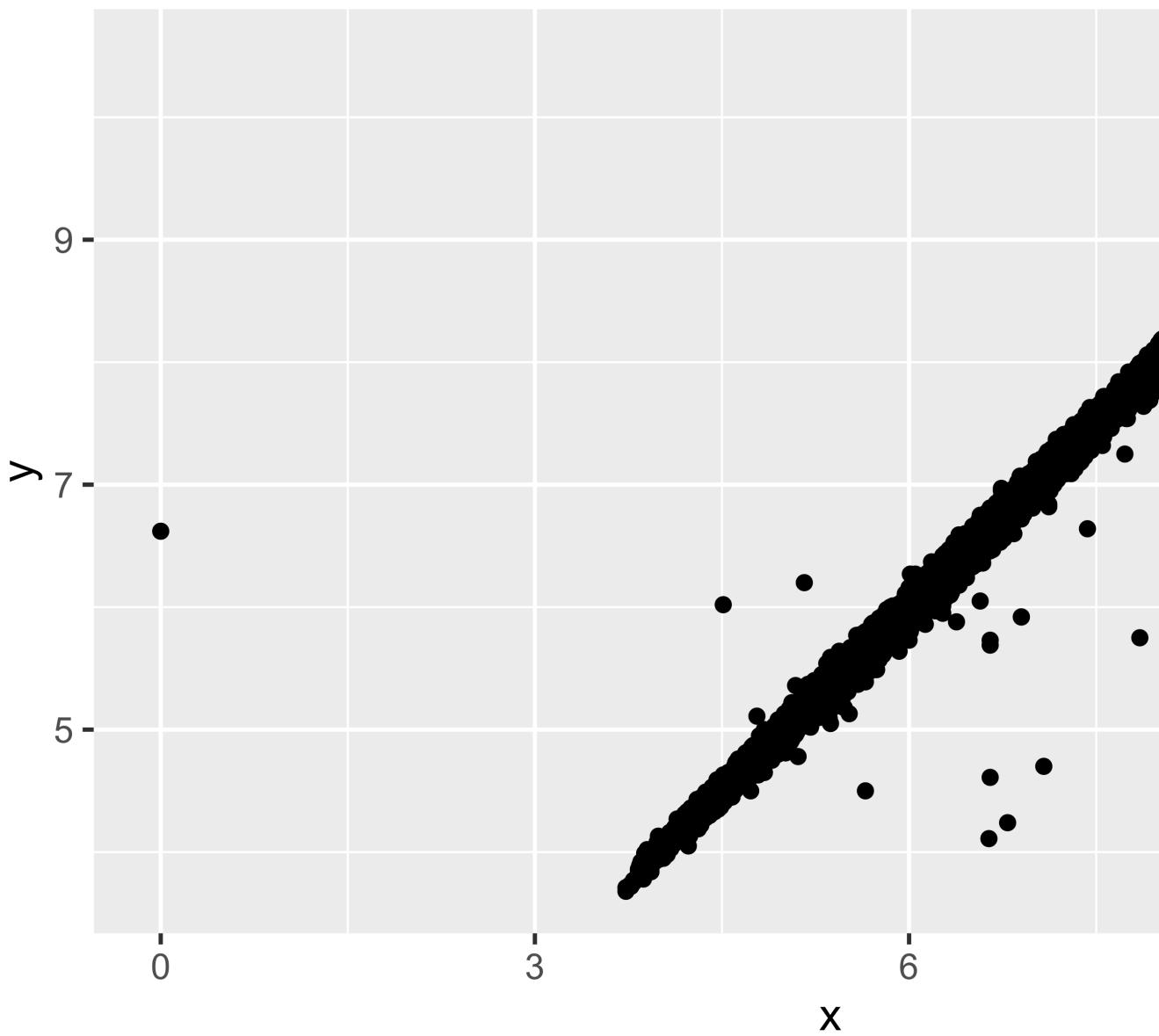
2. 相反，我们建议将异常值替换为缺失值。最简单的方法是使用 `mutate()` 来替换变量的一个修改后的副本。你可以使用 `if_else()` 函数将异常值替换为 `NA`:

```
diamonds2 <- diamonds |>  
mutate(y = if_else(y < 3 | y > 20, NA, y))
```

在图中绘制缺失值并不合理，因此 `ggplot2` 不会将它们包含在图中，但会发出警告，说明这些值已被移除：

```
ggplot(diamonds2, aes(x = x, y = y)) +  
  geom_point()  
#> Warning: Removed 9 rows containing missing values or values outside the scale  
#>  (`geom_point()`).
```

#### 10.4 异常值



299

## 10 探索性数据分析

要阻止该警告，请设置 `na.rm = TRUE`:

```
ggplot(diamonds2, aes(x = x, y = y)) +
  geom_point(na.rm = TRUE)
```

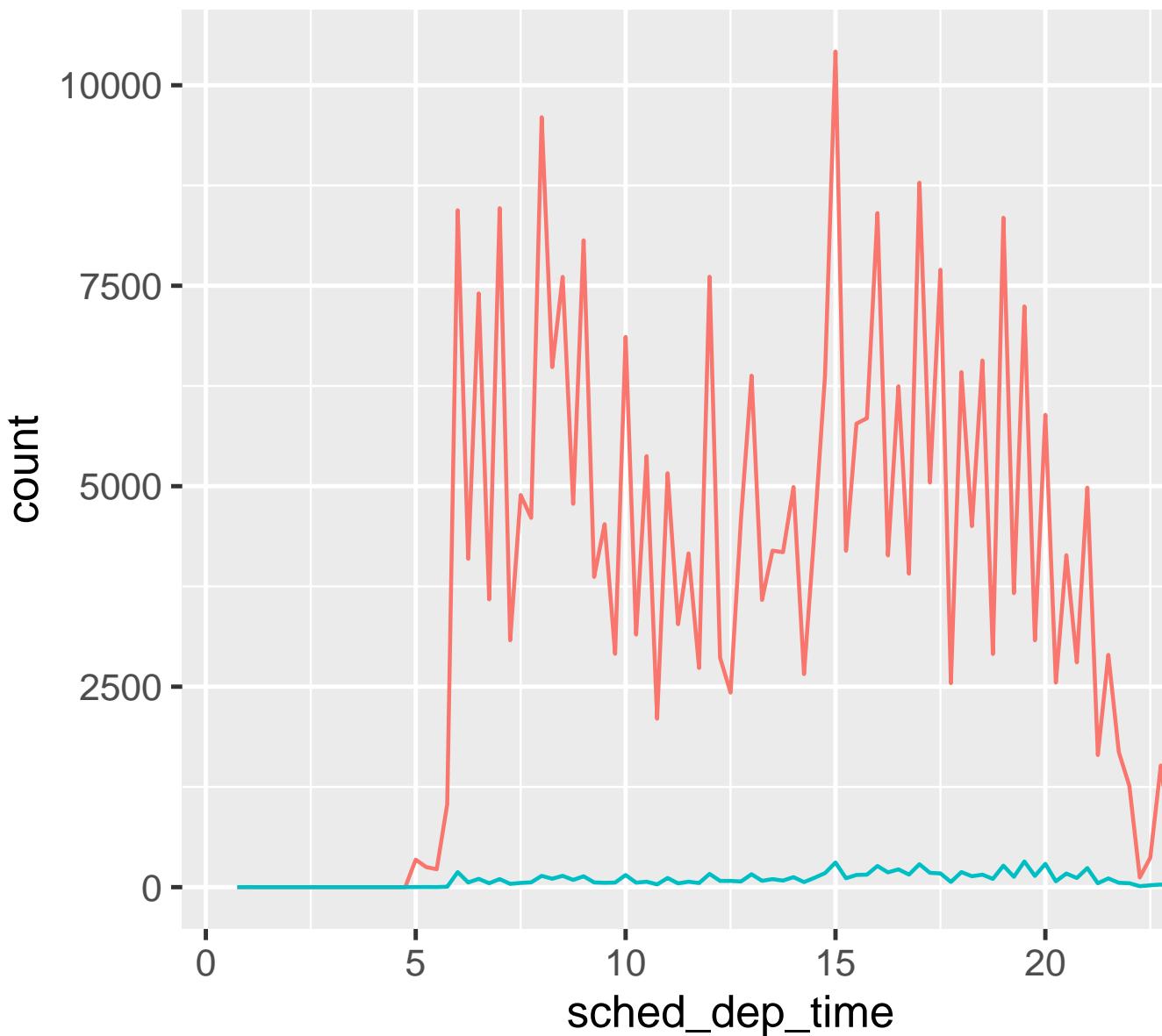
有时你想了解包含缺失值的观测与包含记录值的观测之间有何不同。例如，在 `nycflights13::flights`<sup>1</sup> 数据集中，`dep_time` 变量中的缺失值表示航班被取消。因此，你可能想要比较取消航班和未取消航班的计划起飞时间。可以通过创建一个新变量来实现这一点，使用 `is.na()` 函数来检查 `dep_time` 是否缺失。

```
nycflights13::flights |>
  mutate(
    cancelled = is.na(dep_time),
    sched_hour = sched_dep_time %/% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + (sched_min / 60)
  ) |>
  ggplot(aes(x = sched_dep_time)) +
  geom_freqpoly(aes(color = cancelled), binwidth = 1/4)
```

---

<sup>1</sup>记住，当我们需要明确指出一个函数（或数据集）来自哪个包时，我们会使用特殊的形式 `package::function()` 或 `package::dataset`。

#### 10.4 异常值



然而这个图并不太好，因为未取消的航班比已取消的航班多得多。在下一节中，我们将探索一些技术来改进这种比较。

#### 10.4.1 练习

1. 直方图中缺失值会发什么？条形图中缺失值会发什么？为什么直方图和条形图中缺失值的处理方式不同？
2. 在 `mean()` 和 `sum()` 函数中，`na.rm = TRUE` 的作用是什么？
3. 重新创建按航班是否被取消着色的 `scheduled_dep_time` 的频数图。同时根据 `cancelled` 变量进行分面。在分面函数中尝试使用 `scales` 变量的不同值，以减轻未取消航班多于取消航班的影响。

## 10.5 协变 (covariation)

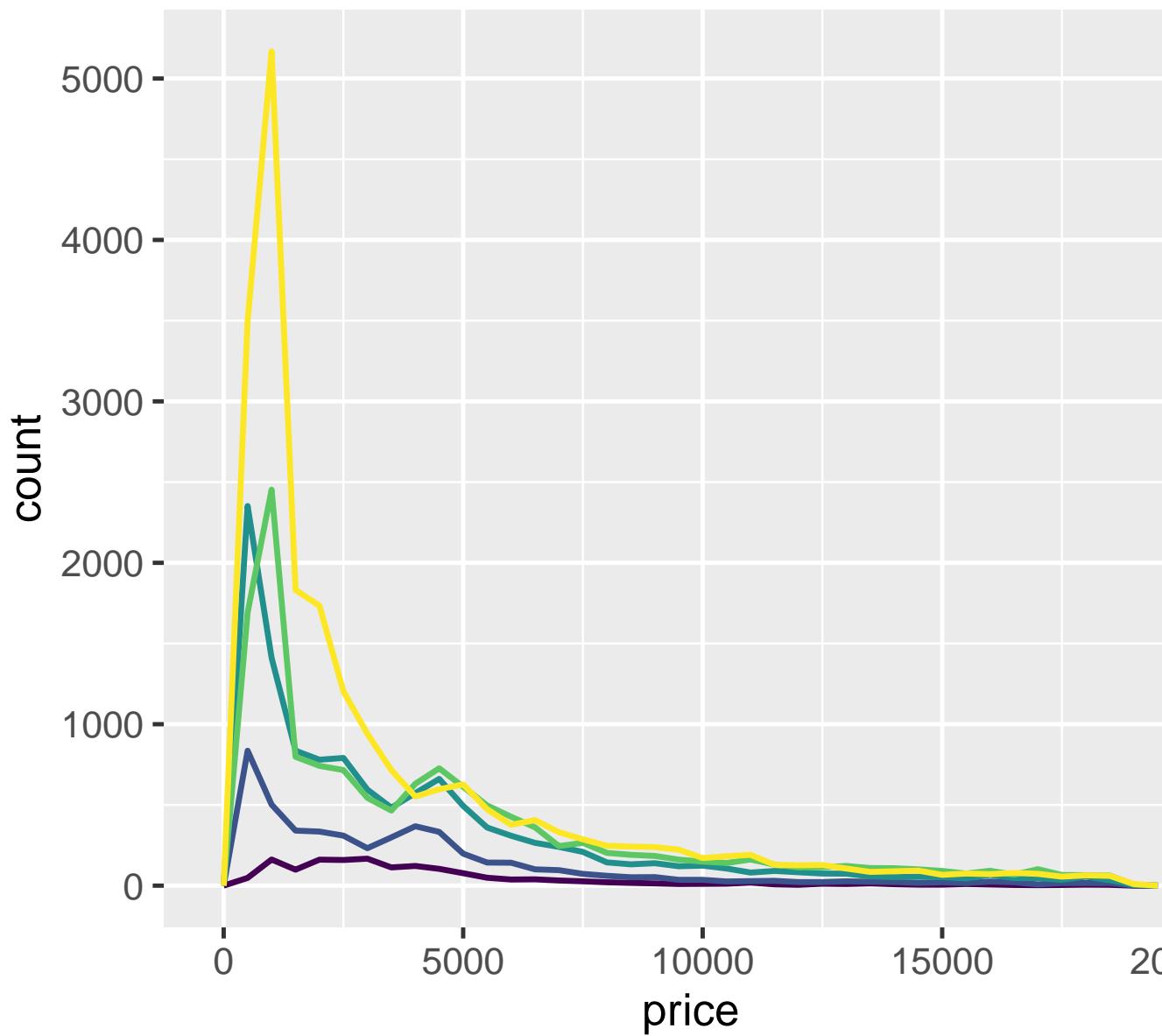
如果变异描述了一个变量内部的行为，那么协变描述了变量之间的行为。协变是两个或多个变量的值以相关的方式一起变化的趋势。发现协变的最佳方法是可视化两个或多个变量之间的关系。

#### 10.5.1 一个分类变量和一个数值变量

例如，使用 `geom_freqpoly()` 来探索钻石的价格如何随其质量 (`cut`) 的变化而变化：

```
ggplot(diamonds, aes(x = price)) +  
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)
```

## 10.5 协变 (covariation)



## 10 探索性数据分析

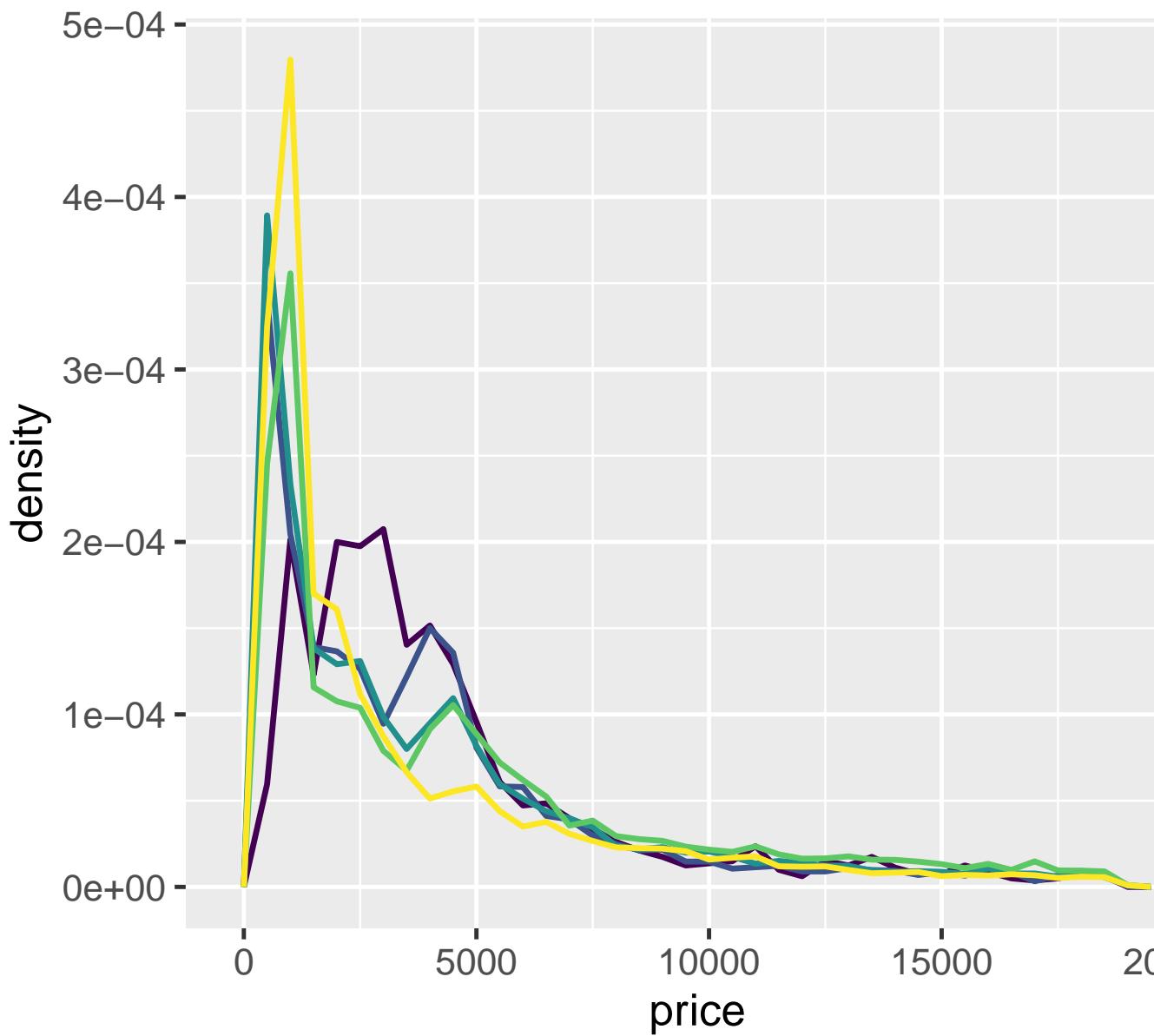
请注意，`ggplot2` 为 `cut` 使用了有序的颜色刻度，因为它在数据中定义为有序因子变量。你将在小节 ?? 中学到更多关于它的知识。

`geom_freqpoly()` 的默认外观在这里并不那么有用，因为由总计数确定的高度在各个 `cut` 之间差异很大，使得难以看出它们分布形状的差异。

为了使比较更容易，我们需要交换 y 轴上的显示内容。我们不显示计数，而是显示密度（density），密度是标准化的计数，使得每个频率多边形的面积都为 1。

```
ggplot(diamonds, aes(x = price, y = after_stat(density))) +  
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)
```

10.5 协变 (covariation)



305

## 10 探索性数据分析

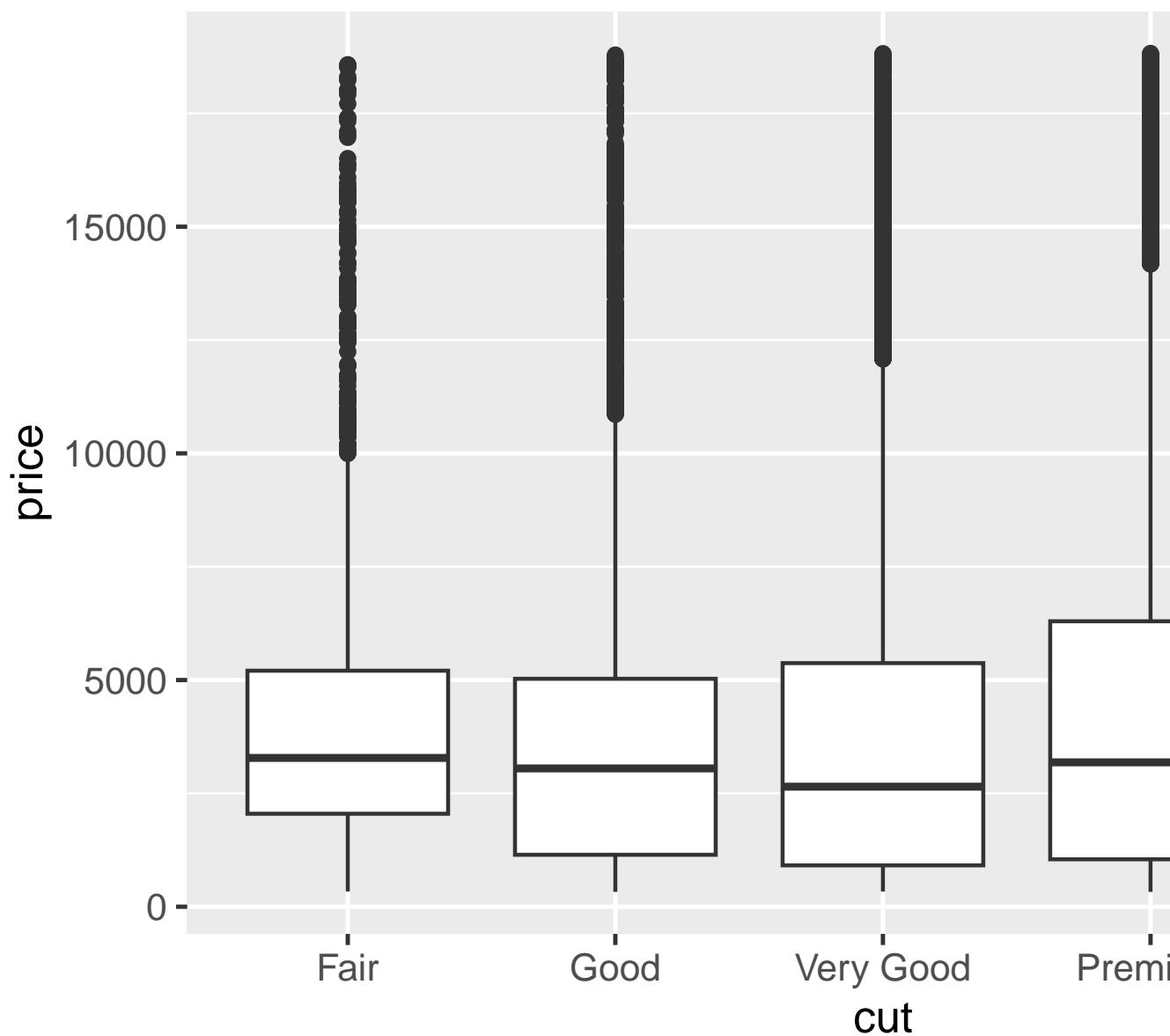
请注意，我们正在将 `density` 映射到 `y` 轴上，但由于 `density` 不是 `diamonds` 数据集中的变量，首先需要计算它。我们使用 `after_stat()` 函数来完成这一操作。

这个图有一个相当令人惊讶的地方——看起来质量一般的钻石（即最低质量）的平均价格最高！但也许这是因为频数多边形图有点难以解读，这个图中有很多内容。

探索这种关系的一种视觉上更简单的方法是使用并排箱线图。

```
ggplot(diamonds, aes(x = cut, y = price)) +  
  geom_boxplot()
```

10.5 协变 (covariation)



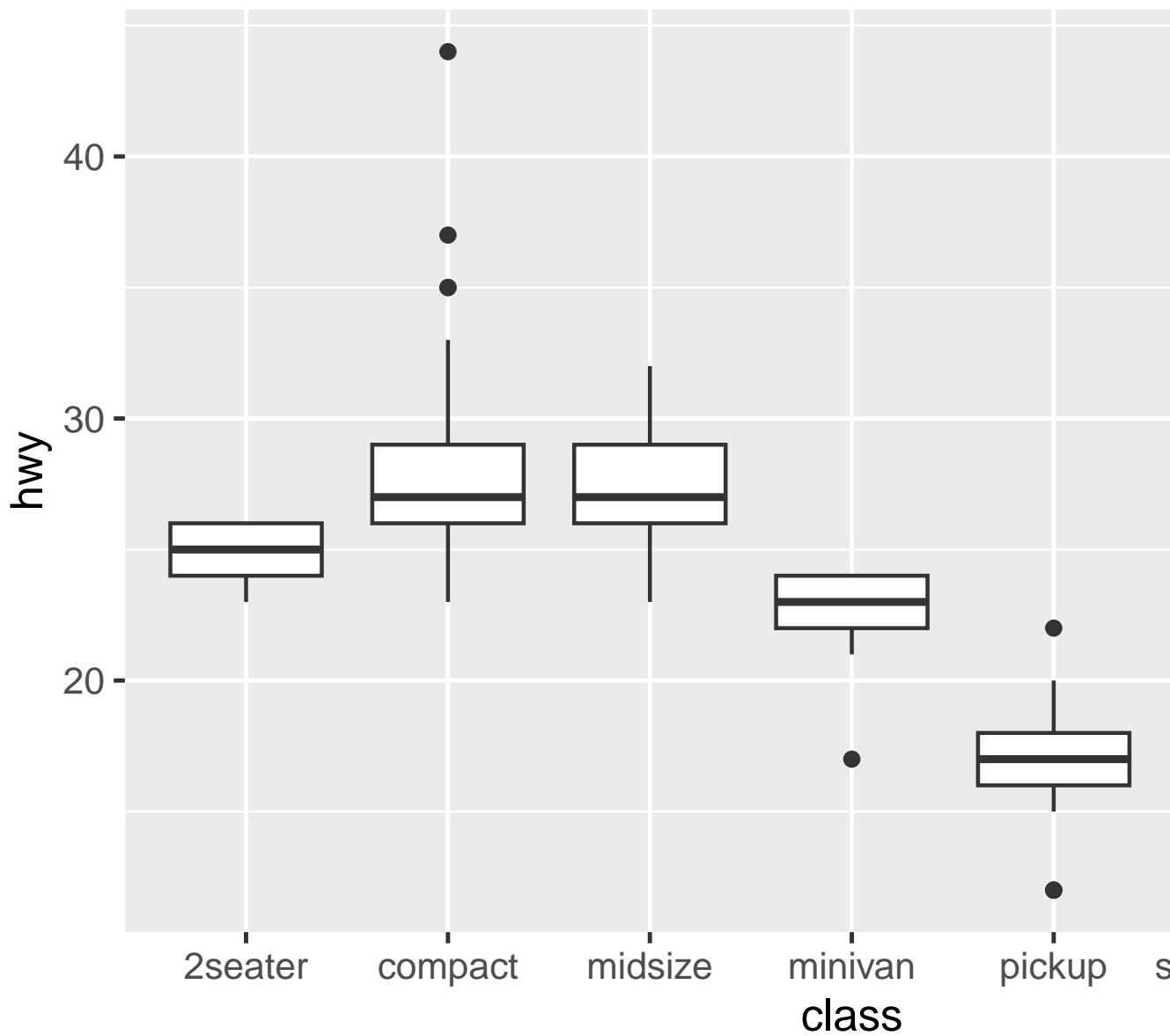
## 10 探索性数据分析

关于分布的信息我们看到的要少得多，但箱线图更加紧凑，因此我们可以更容易地比较它们（并且可以在一个图上展示更多）。这支持了一个反直觉的发现，即更高质量的钻石通常更便宜！在练习中，你将面临的挑战是找出为什么。

`cut` 是一个有序因子：fair（一般）比 good（良好）差，good（良好）比 very good（很好）差，依此类推。许多分类变量并没有这样的内在顺序，因此你可能希望重新排序它们以创建更具信息量的显示。一种方法是使用 `fct_reorder()` 函数，你将在小节 `??` 中了解该函数的更多信息，但因为我们觉得它非常有用，所以想在这里给你一个快速的预览。例如，考虑 `mpg` 数据集中的 `class` 变量。你可能想知道不同类别之间的公路里程是如何变化的：

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

10.5 协变 (covariation)

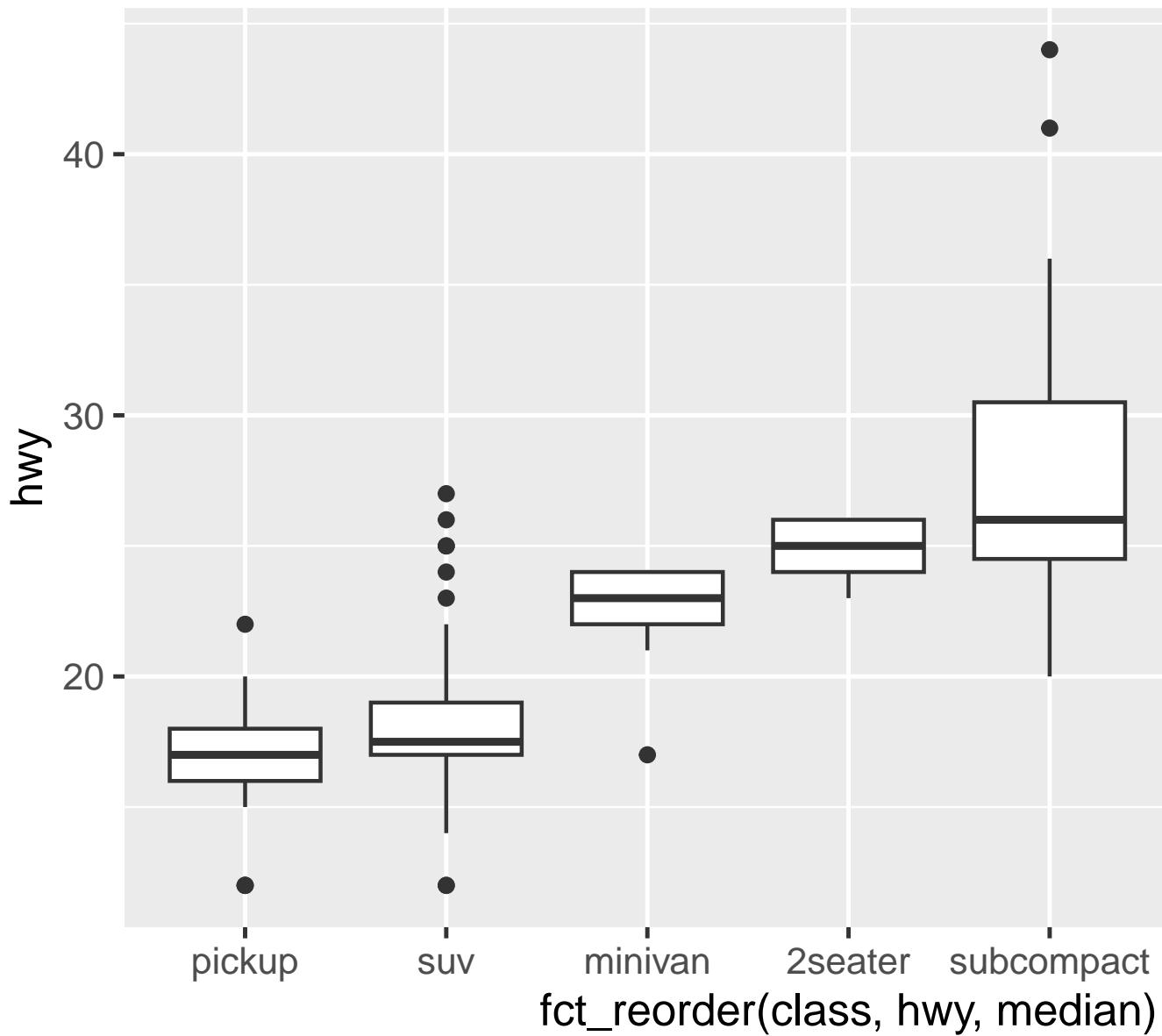


## 10 探索性数据分析

为了使趋势更容易看到，我们根据 `hwy` 的中位数对 `class` 重新排序：

```
ggplot(mpg, aes(x = fct_reorder(class, hwy, median), y = hwy)) +  
  geom_boxplot()
```

10.5 协变 (covariation)

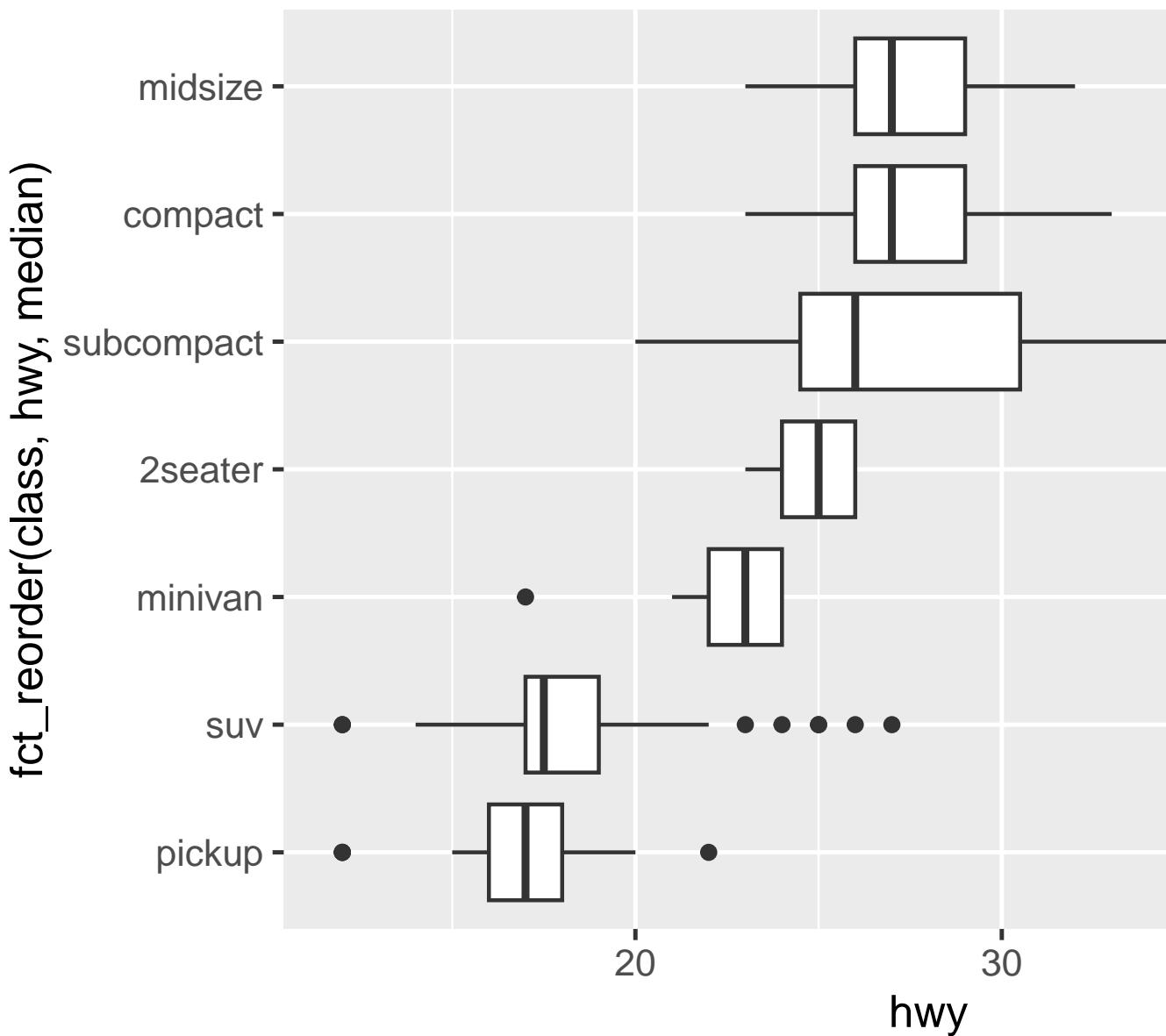


## 10 探索性数据分析

如果你的变量名很长，那么将 `geom_boxplot()` 旋转  $90^\circ$  将会更易于阅读。你可以通过交换 x 和 y 的美学映射来实现这一点。

```
ggplot(mpg, aes(x = hwy, y = fct_reorder(class, hwy, median))) +  
  geom_boxplot()
```

10.5 协变 (covariation)



### 10.5.1.1 练习

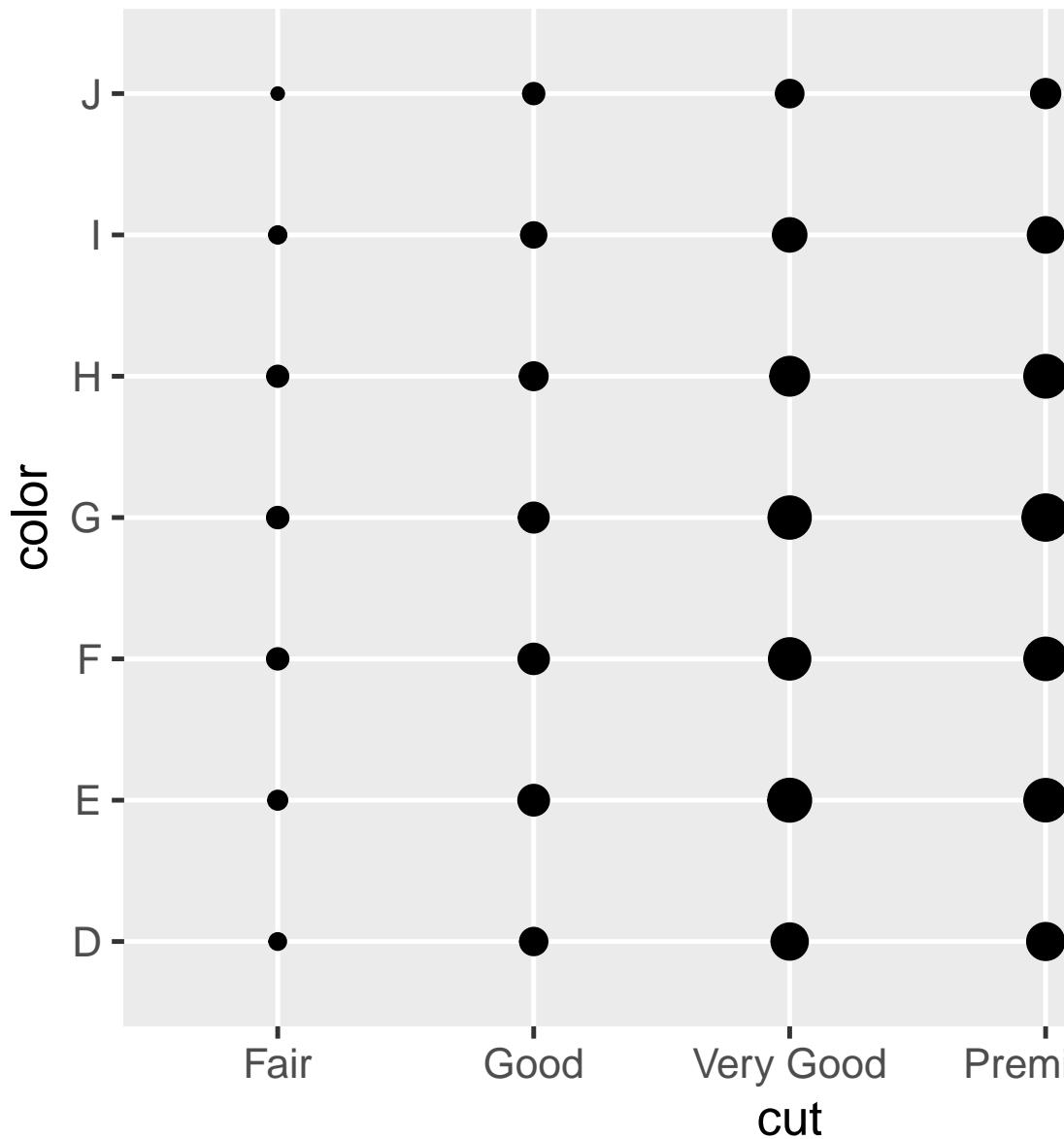
1. 利用你学到的知识来改进取消航班与非取消航班起飞时间的可视化。
2. 基于 EDA, diamonds 数据集中哪个变量看起来对预测钻石的价格最重要? 这个变量与切割方式 (`cut`) 是如何相关的? 为什么这两个关系的组合会导致低质量的钻石更昂贵?
3. 不交换 x 和 y 变量, 而是在垂直箱线图中添加 `coord_flip()` 作为新层来创建水平箱线图。这与交换变量有何不同?
4. 箱线图的一个问题是它们在数据量较小的时期开发, 往往会显示过多的“异常值”。一种解决这个问题的方法是使用字母值图。安装 `lvplot` 包, 并尝试使用 `geom_lv()` 来显示 `price` 与 `cut` 的分布。你学到了什么? 你如何解释这些图?
5. 使用 `geom_violin()` 创建钻石价格与钻石数据集中一个分类变量的可视化, 然后使用分面的 `geom_histogram()`, 然后是着色的 `geom_freqpoly()`, 最后是着色的 `geom_density()`。比较和对比这四种图。根据分类变量的水平可视化数值变量分布的每种方法各有什么优缺点?
6. 如果你有一个小型数据集, 有时使用 `geom_jitter()` 来避免过度绘图是很有用的, 这样可以更容易地看到连续变量和分类变量之间的关系。`gbeeswarm` 包提供了多种与 `geom_jitter()` 类似的方法。列出它们并简要描述每种方法的作用。

## 10.5 协变 (covariation)

### 10.5.2 两个分类变量

为了可视化分类变量之间的协变关系，你需要计算这些分类变量每个水平组合的观察数。一种方法是依赖内置的 `geom_count()` 函数：

```
ggplot(diamonds, aes(x = cut, y = color)) +  
  geom_count()
```



## 10.5 协变 (covariation)

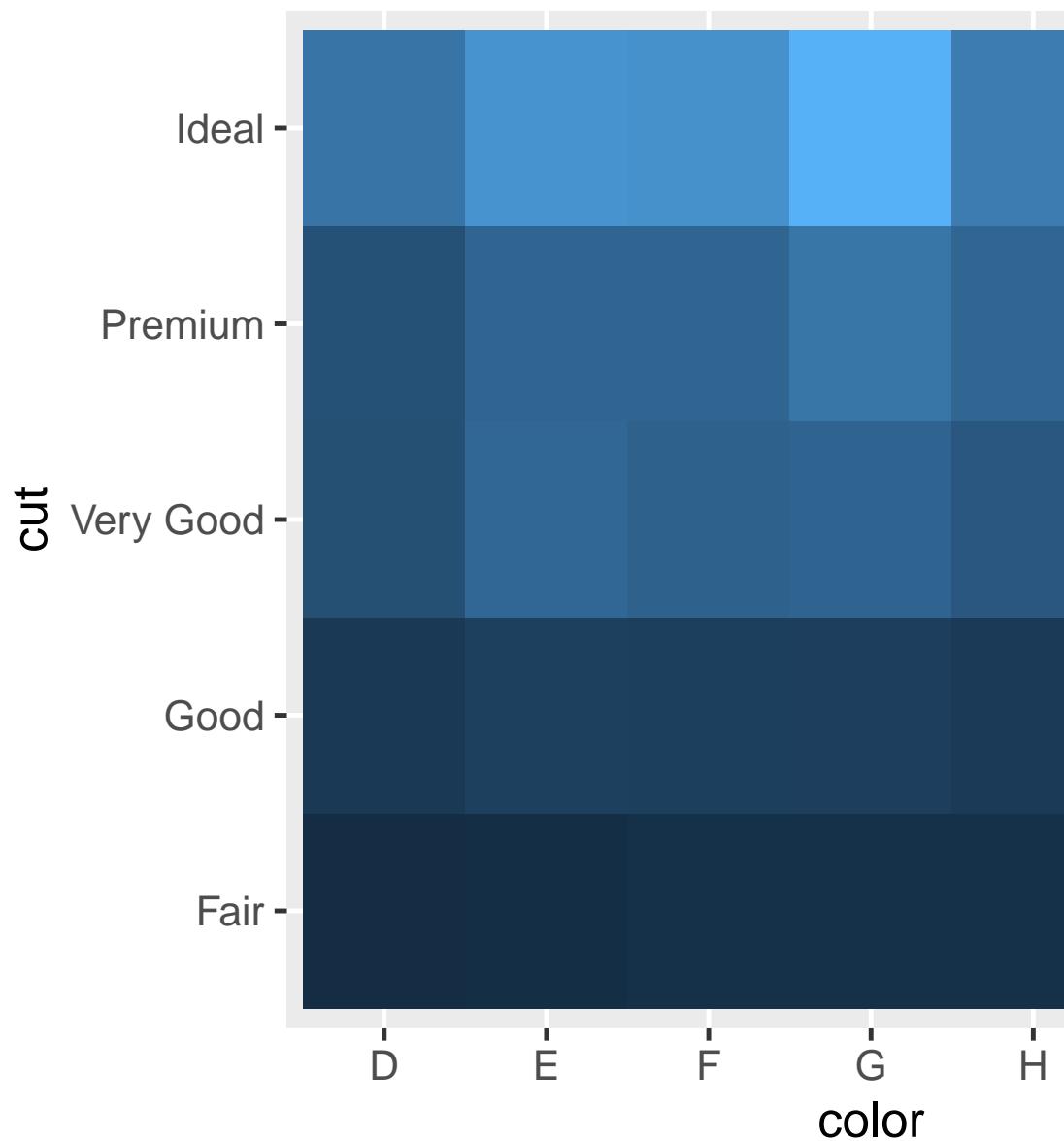
图中的每个圆圈的大小显示了每个值组合出现了多少次观测。协变将表现为特定 x 值和特定 y 值之间的强相关性。

探索这些变量之间关系的另一种方法是使用 dplyr 来计算计数：

```
diamonds |>
  count(color, cut)
#> # A tibble: 35 x 3
#>   color    cut       n
#>   <ord> <ord>     <int>
#> 1 D      Fair      163
#> 2 D      Good      662
#> 3 D      Very Good 1513
#> 4 D      Premium   1603
#> 5 D      Ideal     2834
#> 6 E      Fair      224
#> # i 29 more rows
```

然后使用 `geom_tile()` 和填充美学进行可视化：

```
diamonds |>
  count(color, cut) |>
  ggplot(aes(x = color, y = cut)) +
  geom_tile(aes(fill = n))
```



## 10.5 协变 (covariation)

如果分类变量是无序的，你可能想要使用 `seriation` 包来同时重新排序行和列，以便更清楚地揭示有趣的模式。对于较大的图形，你可能想要尝试 `heatmaply` 包，它创建交互式图形。

### 10.5.2.1 练习

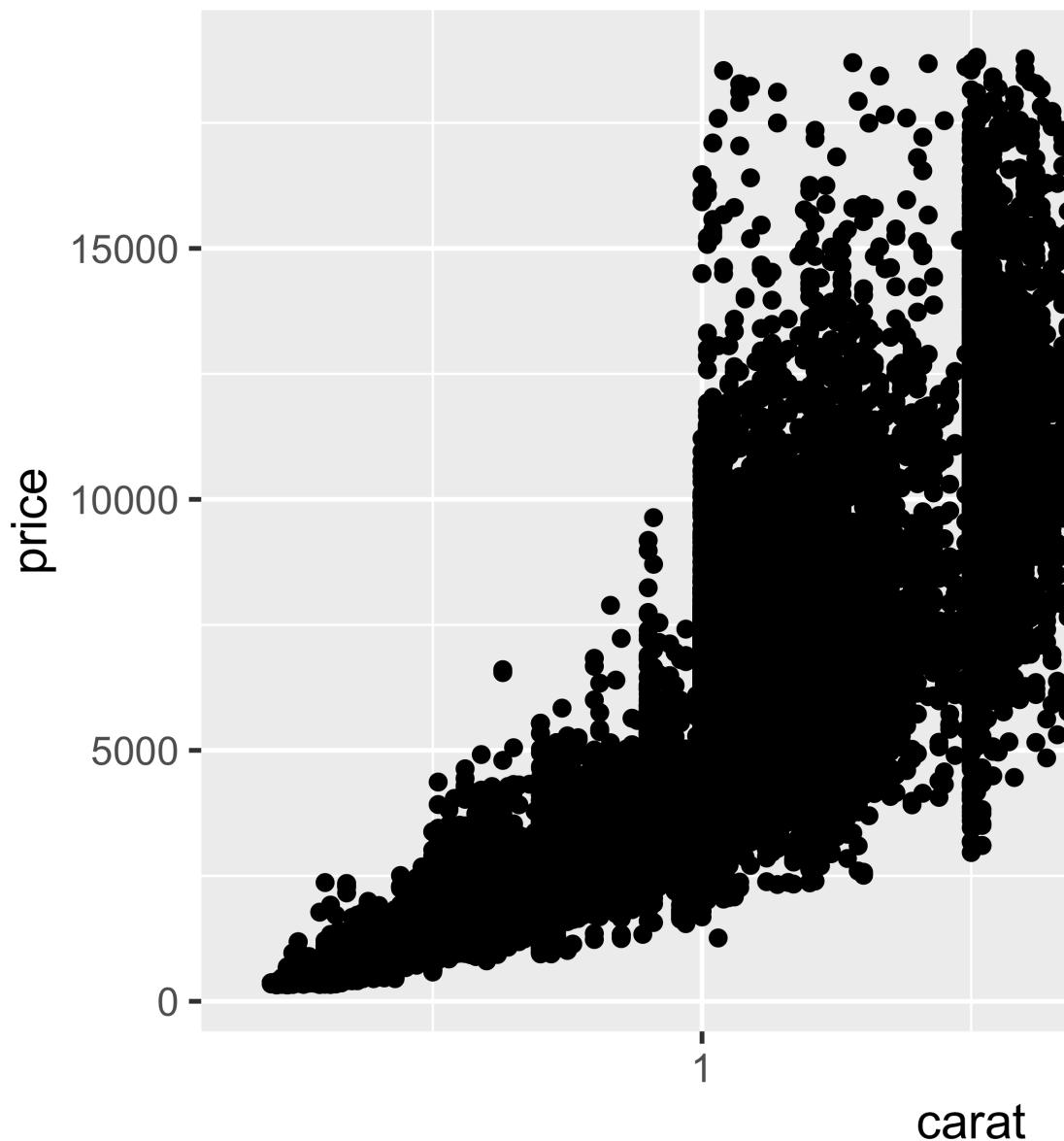
1. 如何重新缩放上面的计数数据集，以更清楚地显示 `color` 中的 `cut` 分布，或 `cut` 中的 `color` 分布？
2. 如果 `color` 映射到 `x` 美学，`cut` 映射到 `fill` 美学，使用分段条形图你能得到哪些不同的数据见解？计算每个分段中的计数。
3. 使用 `geom_tile()` 结合 `dplyr` 来探索平均航班起飞延误如何因目的地和年份月份而异。这个图为什么难以阅读？你如何改进它？

### 10.5.3 两个数值变量

你已经看到了可视化两个数值变量之间协变关系的一种好方法：使用 `geom_point()` 绘制散点图。你可以在点的模式中看到协变关系。例如，你可以看到钻石的克拉大小与其价格之间的正相关关系：克拉数越多的钻石价格越高。这种关系是呈指数型的。

```
ggplot(smaller, aes(x = carat, y = price)) +  
  geom_point()
```

10 探索性数据分析



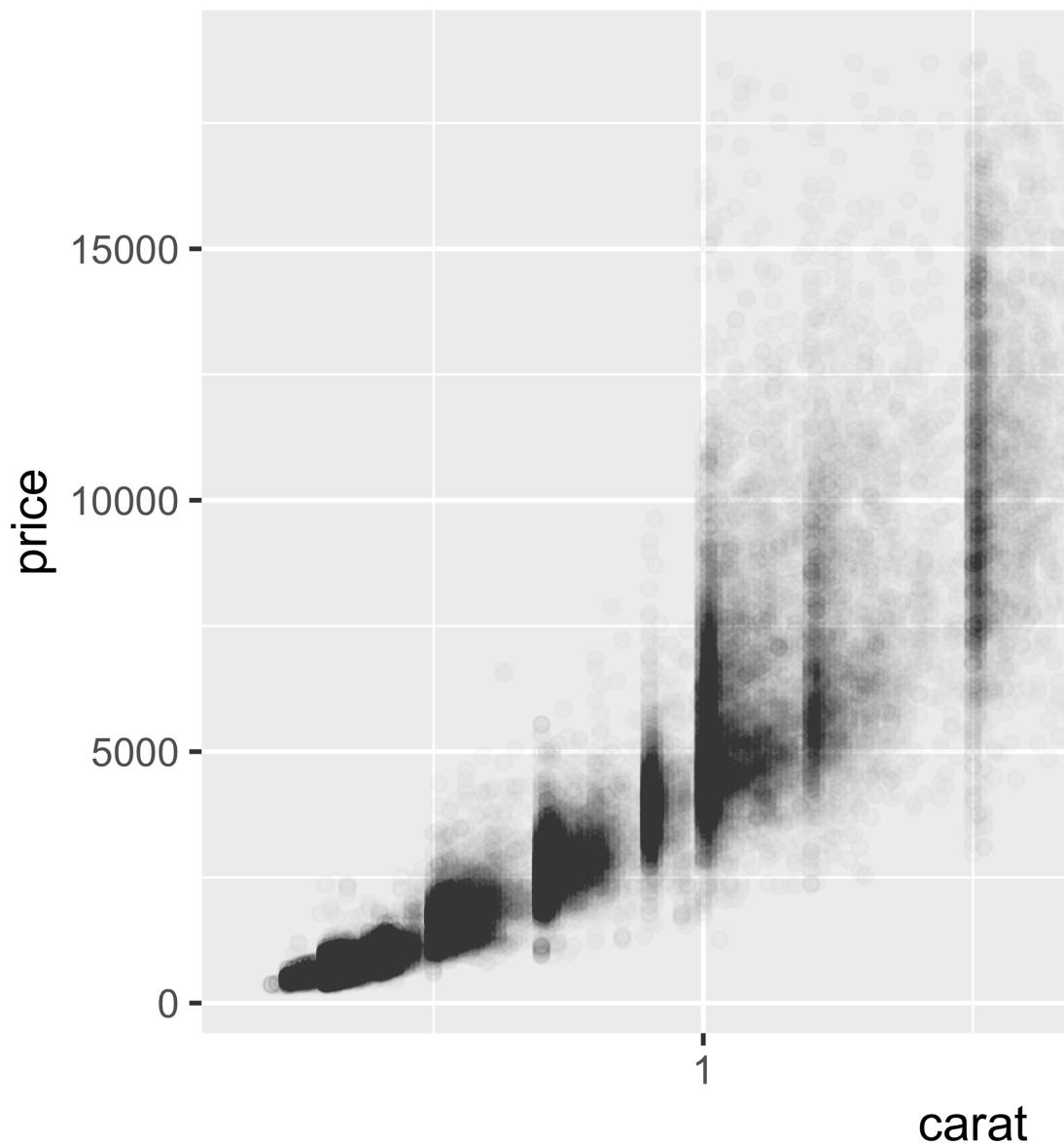
## 10.5 协变 (covariation)

(在本节中，我们将使用 `smaller` 数据集来重点关注克拉数小于 3 的钻石)

随着数据集大小的增加，散点图变得不那么有用，因为点开始重叠并堆积成均匀的黑色区域，这使得难以判断二维空间中数据密度的差异，也难以发现趋势。你已经看到了一种解决问题的方法：使用 `alpha` 美学属性来增加透明度。

```
ggplot(smaller, aes(x = carat, y = price)) +  
  geom_point(alpha = 1 / 100)
```

10 探索性数据分析



## 10.5 协变 (covariation)

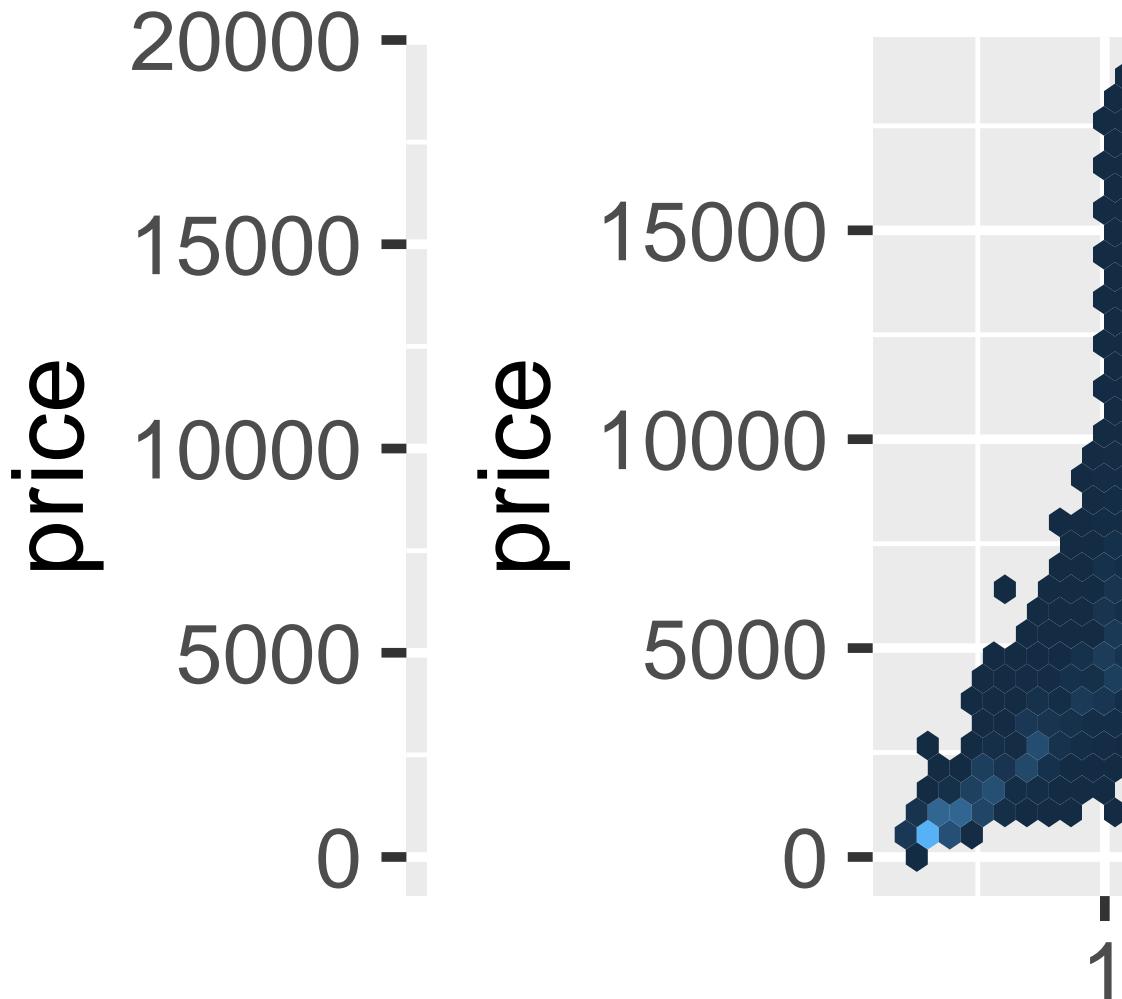
但是，对于非常大的数据集，使用透明度可能会很有挑战性。另一种解决方案是使用分组（bin）。之前您使用 `geom_histogram()` 和 `geom_freqpoly()` 在一维中进行分组。现在您将学习如何使用 `geom_bin2d()` 和 `geom_hex()` 在二维中进行分组。

`geom_bin2d()` 和 `geom_hex()` 将坐标平面划分为二维的分组（bins），然后使用填充颜色来表示每个分组中有多少点。`geom_bin2d()` 创建矩形的分组。`geom_hex()` 创建六边形的分组。要使用 `geom_hex()`，您需要安装 `hexbin` 包。

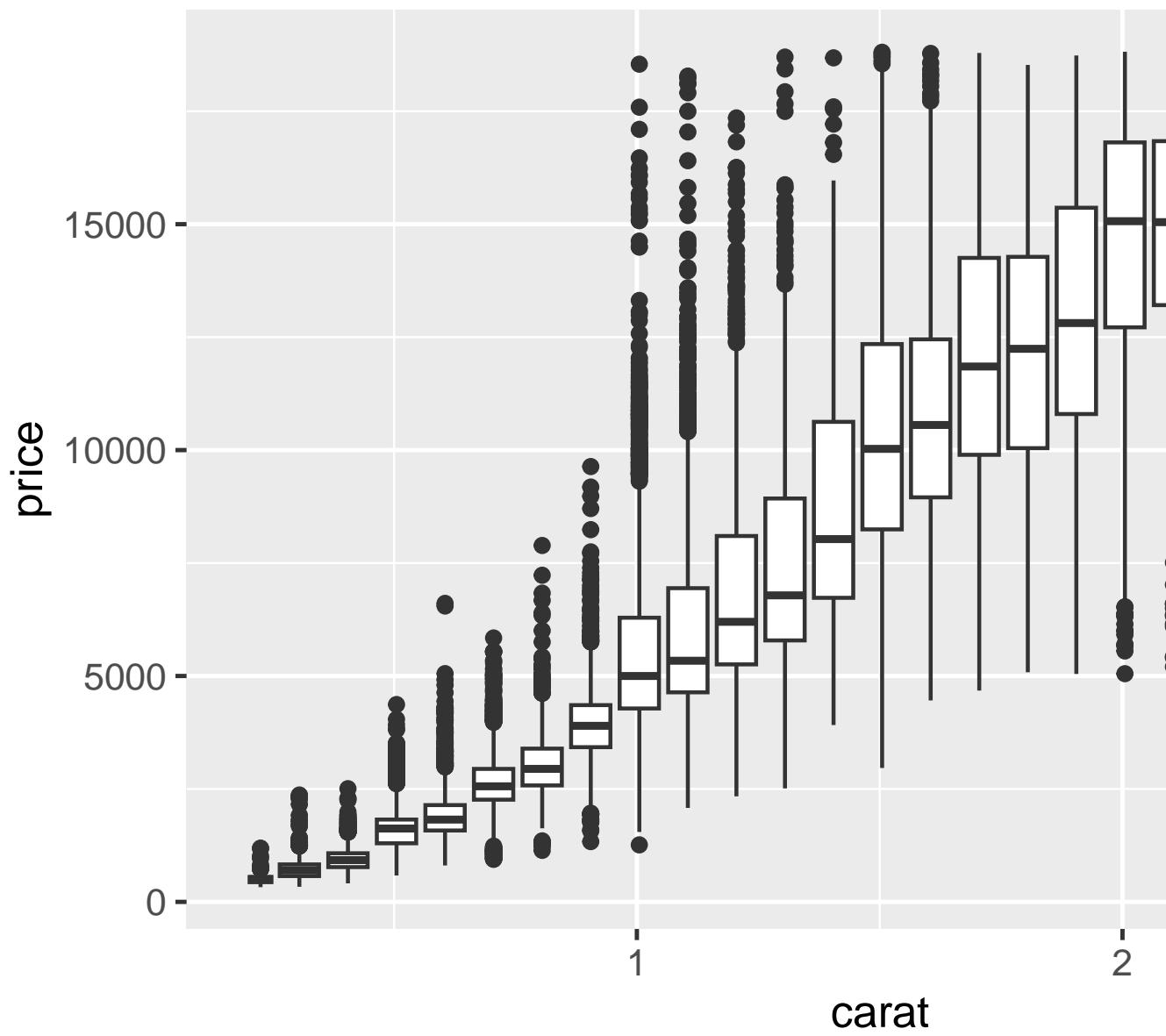
```
ggplot(smaller, aes(x = carat, y = price)) +  
  geom_bin2d()  
  
# install.packages("hexbin")  
ggplot(smaller, aes(x = carat, y = price)) +  
  geom_hex()
```

另一种选择是将一个连续变量进行分组，使其表现得像一个分类变量。然后，您可以使用您学过的用于可视化分类变量和连续变量组合的技术之一。例如，您可以对 `carat` 进行分组，然后为每个组显示一个箱线图：

```
ggplot(smaller, aes(x = carat, y = price)) +  
  geom_boxplot(aes(group = cut_width(carat, 0.1)))
```



10.5 协变 (covariation)



上面使用 `cut_width(x, width)` 将 `x` 分成宽度为 `width` 的分组。默认情况下，无论观测的数量如何，箱线图看起来大致相同（除了异常值的数量），因此很难看出每个箱线图汇总了不同数量的点。一种表示这种差异的方法是让箱线图的宽度与点的数量成比例，这通过设置 `varwidth = TRUE` 来实现。

#### 10.5.3.1 练习

1. 除了使用箱线图来总结条件分布外，你还可以使用频数多边形。在使用 `cut_width()` 和 `cut_number()` 时需要考虑什么？这对可视化 `carat` 和 `price` 的二维分布有什么影响？
2. 可视化根据 `price` 划分的 `carat` 分布。
3. 非常大的钻石的价格分布与小钻石的价格分布相比如何？是否符合你的预期，还是让你感到惊讶？
4. 结合你学过的两种技术来可视化 `cut`、`carat` 和 `price` 的联合分布。
5. 二维图可以揭示一维图中不可见的异常值。例如，以下图中的某些点具有不寻常的 `x` 和 `y` 值组合，即使分别查看这些点的 `x` 和 `y` 值时看起来是正常的，也使得这些点成为异常值。为什么在这种情况下散点图是比分组图更好的展示方式？

```
diamonds |>
  filter(x >= 4) |>
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  coord_cartesian(xlim = c(4, 11), ylim = c(4, 11))
```

6. 使用 `cut_number()` 创建包含大致相等数量点的分组，而不是使用 `cut_width()` 创建等宽度的分组。这种方法的优点和缺点是什么？

```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_boxplot(aes(group = cut_number(carat, 20)))
```

## 10.6 模式与模型

如果两个变量之间存在系统关系，它将在数据中呈现为一种模式。如果你发现一个模式，问问自己：

- 这个模式是否可能是巧合（即随机机会）？
- 你如何描述这个模式所暗示的关系？
- 这个模式所暗示的关系有多强？
- 还有哪些其他变量可能会影响这种关系？
- 如果你查看数据的各个子组，这种关系会发生变化吗？

数据中的模式提供了关于关系的线索，即它们揭示了协变。如果你把变化看作是一种产生不确定性的现象，那么协变就是一种减少这种不确定性的现象。如果两个变量协变，你可以利用一个变量的值来更好地预测第二个变量的值。如果协变是由因果关系（一个特殊情况）引起的，那么你可以用一个变量的值来控制第二个变量的值。

模型是从数据中提取模式的工具。例如，考虑钻石数据，很难理解切工和价格之间的关系，因为切工和克拉数，以及克拉数和价格之间存在紧密关系。我们可以使用模型来消除价格和克拉数之间的非常强的关系，以便我们可以探索剩余的细微差别。以下代码拟合了一个从 `carat` 预测 `price` 的模型，然后计算残差（预测值与实际值之间的差异）。一旦去除了克拉数的影响，残差就给了

## 10 探索性数据分析

我们钻石价格的一个视图。请注意，我们不是直接使用 `price` 和 `carat` 的原始值，而是首先对它们进行对数转换，并对对数转换后的值拟合模型。然后，我们将残差进行指数化，以将其重新放入原始价格的尺度上。

```
library(tidymodels)

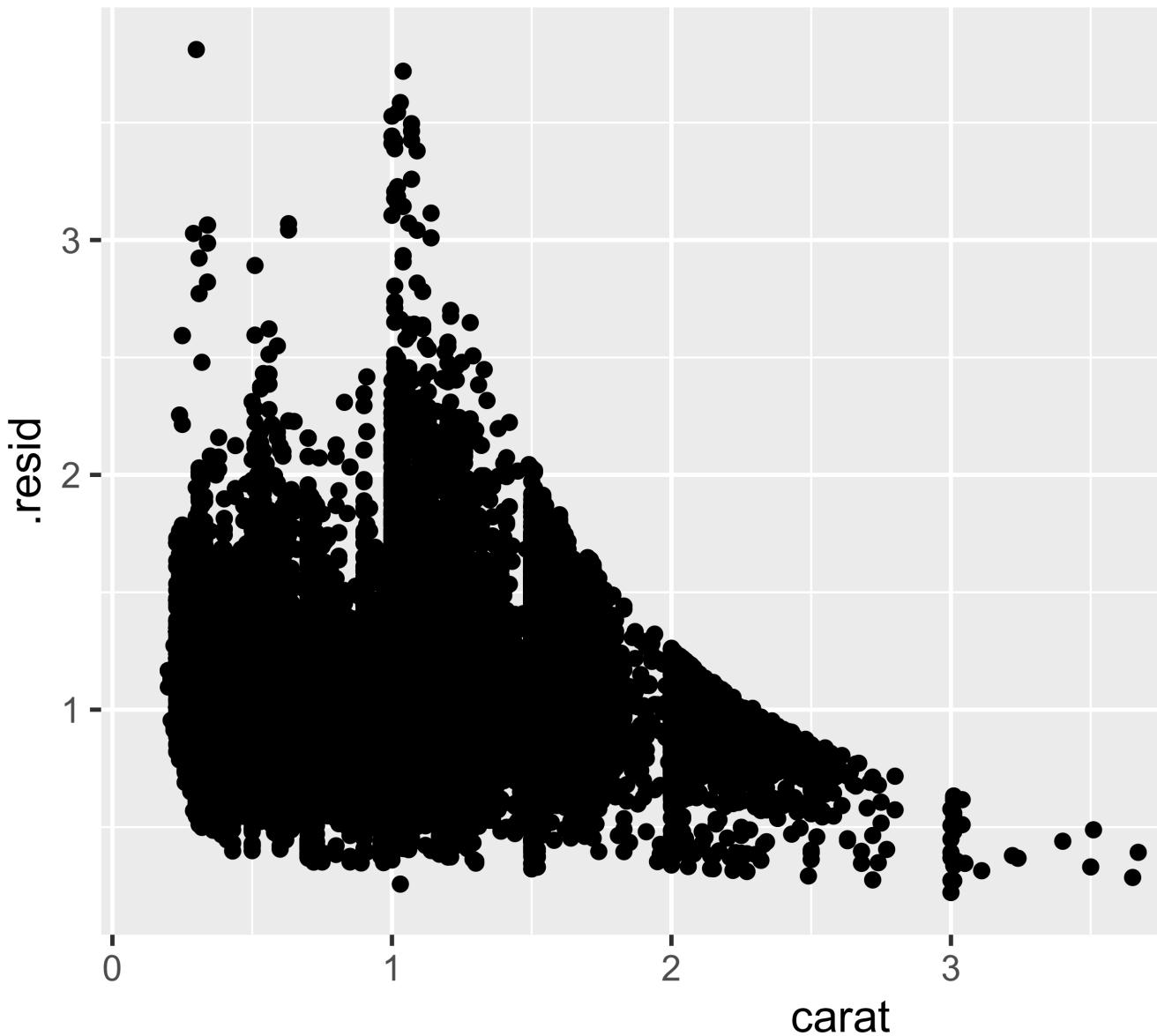
diamonds <- diamonds |>
  mutate(
    log_price = log(price),
    log_carat = log(carat)
  )

diamonds_fit <- linear_reg() |>
  fit(log_price ~ log_carat, data = diamonds)

diamonds_aug <- augment(diamonds_fit, new_data = diamonds) |>
  mutate(.resid = exp(.resid))

ggplot(diamonds_aug, aes(x = carat, y = .resid)) +
  geom_point()
```

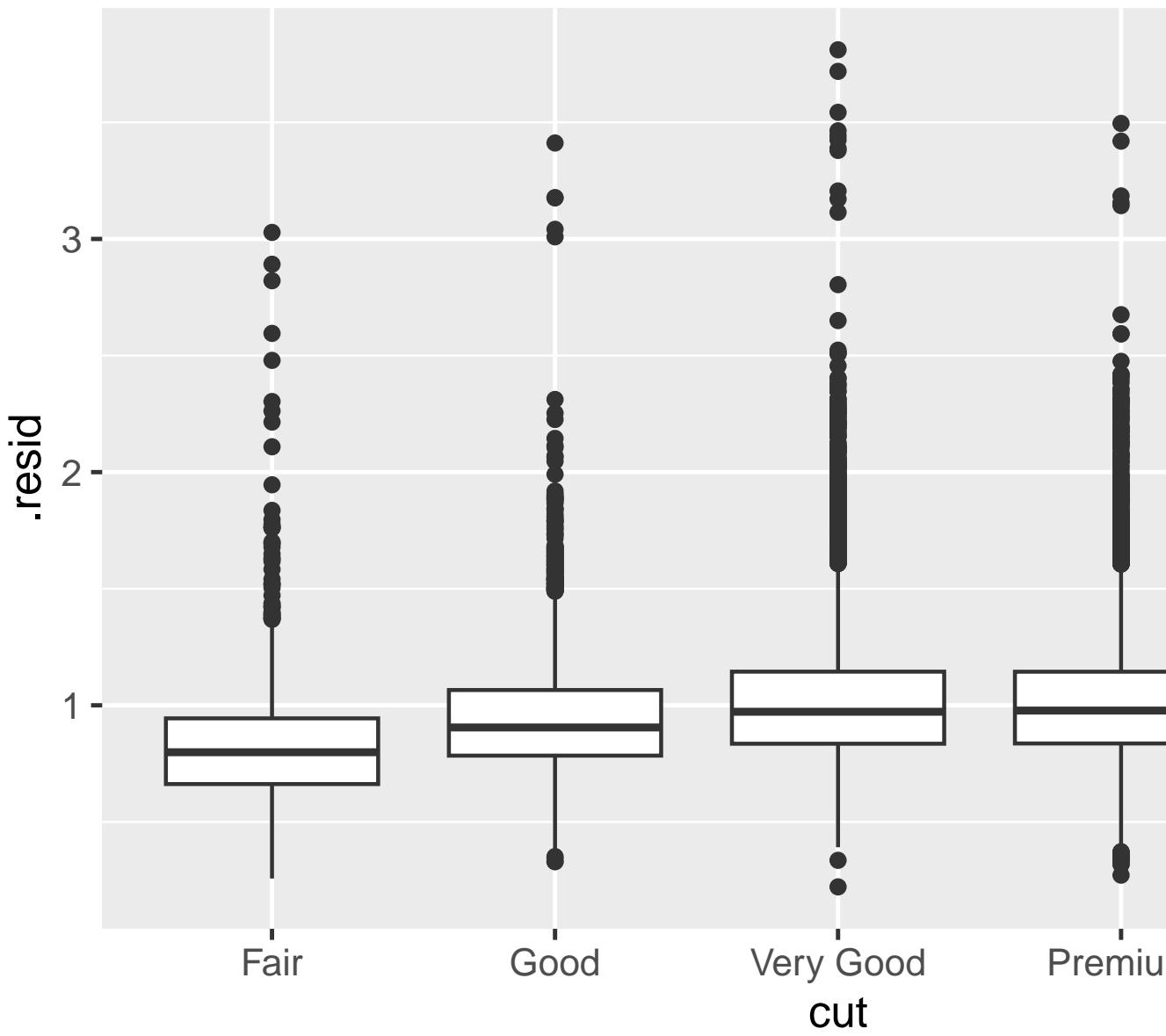
10.6 模式与模型



## 10 探索性数据分析

一旦消除了克拉数和价格之间的强关系，你就可以看到切工和价格之间的关系是你所预期的：相对于它们的大小，质量好的钻石更昂贵。

```
ggplot(diamonds_aug, aes(x = cut, y = .resid)) +  
  geom_boxplot()
```



本书并未讨论建模，因为一旦你掌握了数据处理和编程的工具，理解模型是什么以及它们是如何工作的就会变得更加容易。

## 10.7 小结

在这一章中，你学习了多种工具来帮助你理解数据中的变化。你已经看到了针对单个变量和变量对的技术。如果你的数据中有数十个或数百个变量，这些技术可能看起来非常有限，但它们是所有其他技术的基础。

在下一章中，我们将专注于可以用来交流结果的工具。

# 11 交流

## 11.1 引言

在章节 ?? 中，你学习了如何将图形用作探索工具。当你制作探索性图形时，你甚至在查看之前就知道图中将显示哪些变量。你为了某个目的制作每个图形，可以快速查看它，然后移至下一个图形。在大多数分析过程中，你会生成数十个或数百个图形，其中大多数会立即被丢弃。

现在你理解了你的数据，你需要将你的理解与他人交流。你的听众可能不具备你的背景知识，也不会对数据投入太多关注。为了帮助其他人快速建立对数据的良好心理模型，你需要在使图形尽可能具有自解释性方面投入大量努力。在本章中，你将学习 ggplot2 提供的一些工具来做到这一点。

本章重点关注创建良好图形所需的工具。我们假设你知道你想要什么，只是需要知道如何去做。因此，我们强烈建议将本章与一本优秀的通用可视化书籍结合使用。我们特别喜欢 Albert Cairo 的《[The Truthful Art](#)》。这本书不教授创建可视化的机制，而是专注于你需要考虑的内容以创建有效的图形。

### 11.1.1 必要条件

在本章中我们将再次专注于 ggplot2。我们还将使用 dplyr 进行一些数据操作，使用 scales 来覆盖默认的刻度、标签、变换和调色板，以及一些 ggplot2 的扩展包，包括 Kamil Slowikowski 的 ggrepel (<https://ggrepel.slowkow.com>) 和 Thomas Lin Pedersen 的 patchwork (<https://patchwork.data-imaginist.com>)。如果你还没有这些包，别忘了使用 install.packages() 来安装它们。

```
library(tidyverse)
library(scales)
library(ggrepel)
library(patchwork)
```

## 11.2 标签

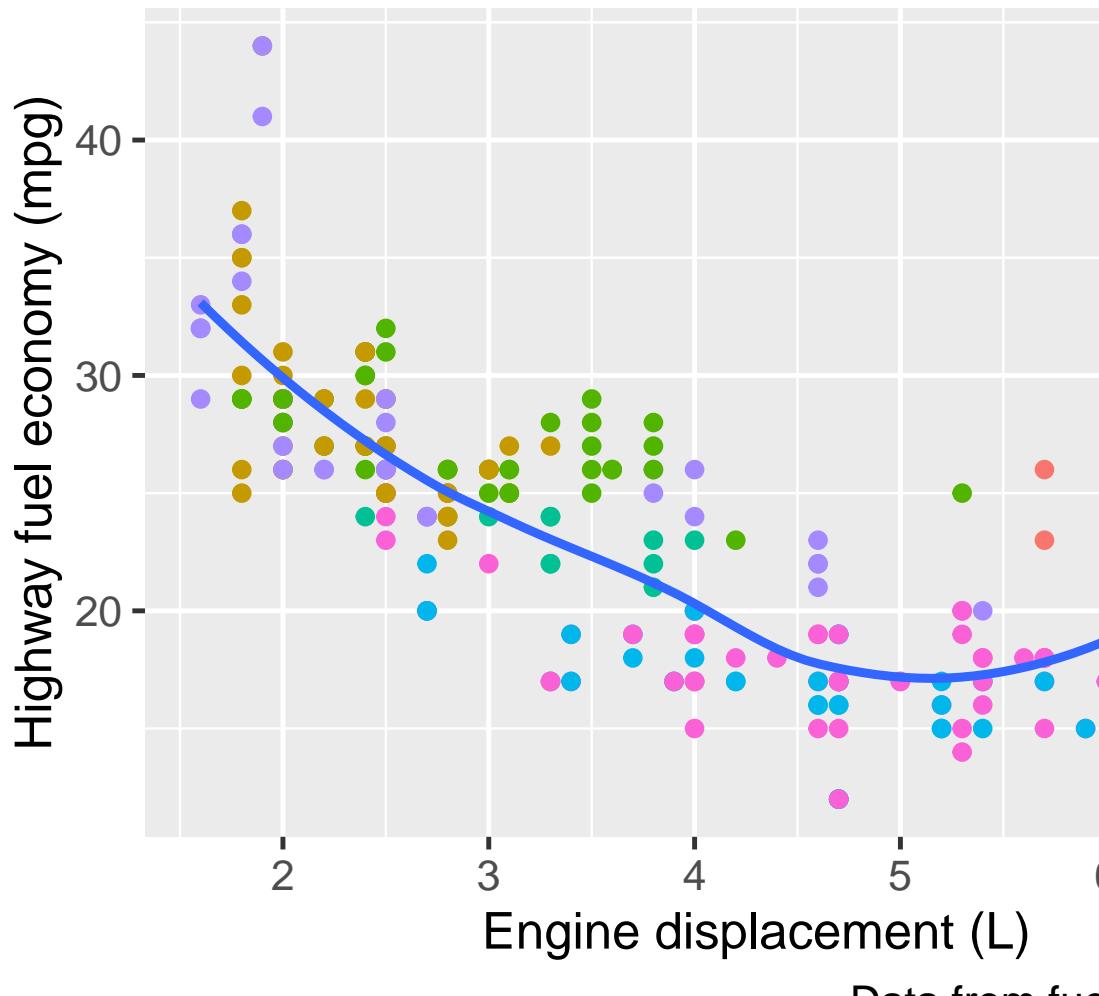
将探索性图形转换为说明性图形的最简单起点是使用好的标签。你可以使用 labs() 函数来添加标签。

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (L)",
    y = "Highway fuel economy (mpg)",
    color = "Car type",
    title = "Fuel efficiency generally decreases with engine size",
```

## 11.2 标签

```
subtitle = "Two seaters (sports cars) are an exception because of their light weight",
caption = "Data from fueleconomy.gov"
)
```

Fuel efficiency generally decreases with engine displacement.  
Two seaters (sports cars) are an exception.



## 11.2 标签

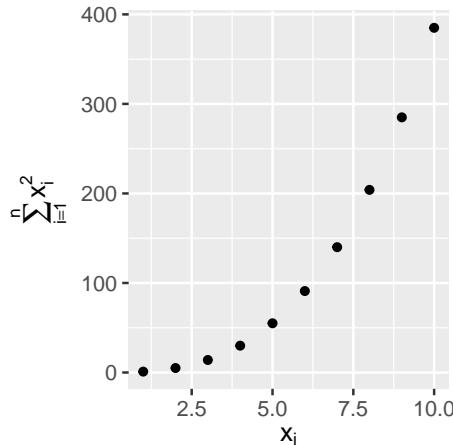
图形的标题 (`title`) 是为了概括主要发现，避免仅描述图形内容的标题，例如“发动机排量和燃油经济性的散点图形”。

如果你需要添加更多文本，还有另外两个有用的标签：副标题 (`subtitle`) 以较小的字体在标题下方添加更多细节，而图注 (`caption`) 在图形的右下角添加文本，通常用于描述数据的来源。你还可以使用 `labs()` 函数来替换轴和图例的标题。通常，最好将简短的变量名替换为更详细的描述，并包含单位。

可以使用数学方程代替文本字符串。只需将"" 替换为 `quote()`，并查阅`?plotmath` 中关于可用选项的信息。

```
df <- tibble(
  x = 1:10,
  y = cumsum(x^2)
)

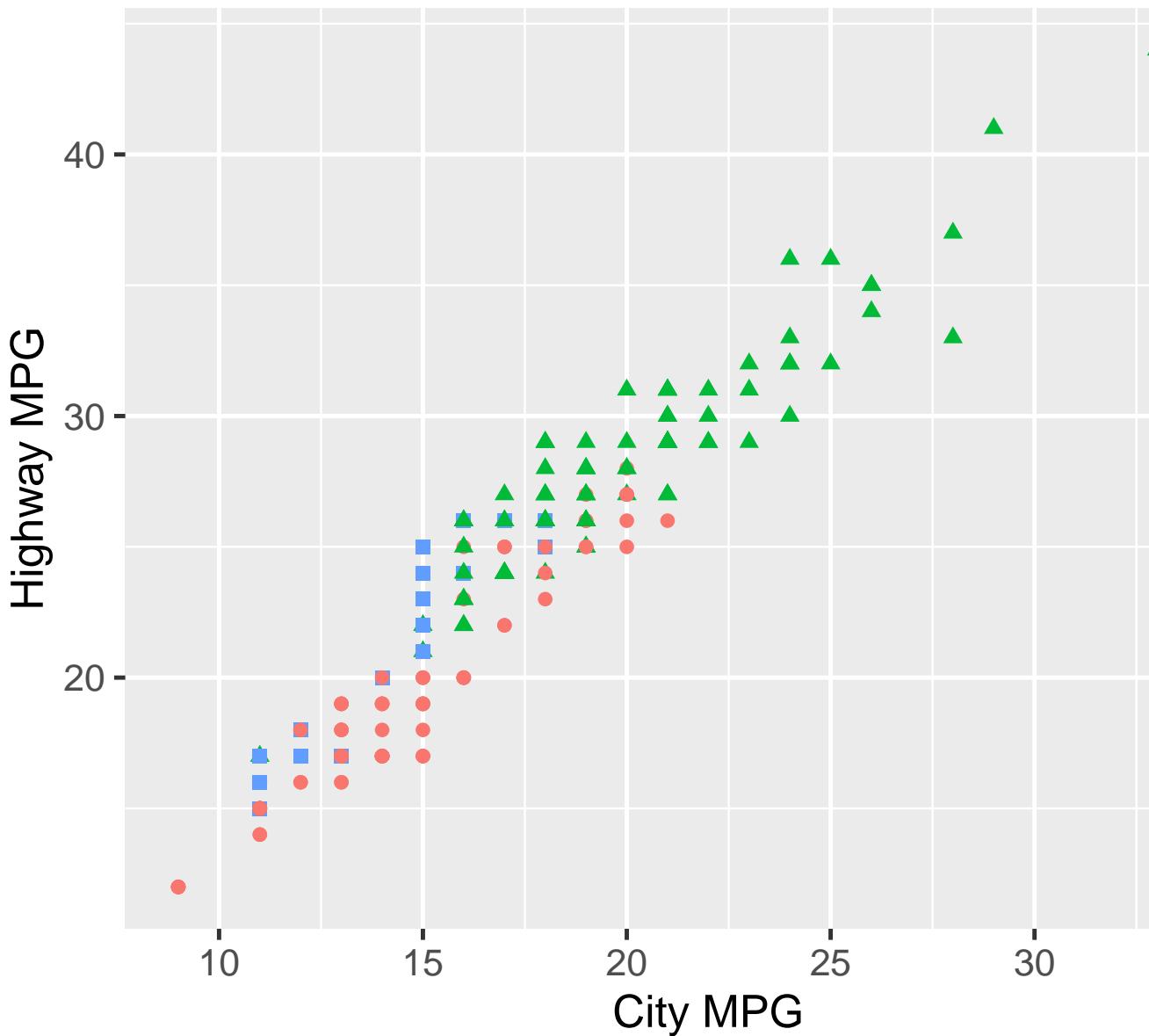
ggplot(df, aes(x, y)) +
  geom_point() +
  labs(
    x = quote(x[i]),
    y = quote(sum(x[i] ^ 2, i == 1, n))
  )
```



### 11.2.1 练习

1. 创建一个包含自定义标题、副标题、图注、x 轴、y 轴和颜色标签的关于燃油经济数据的图形。Create one plot on the fuel economy data with customized title, subtitle, caption, x, y, and color labels.
2. 使用燃油经济数据重新创建以下图形。请注意，点和形状的颜色根据传动系统的类型而变化。

## 11.2 标签



339

3. 选择你在过去一个月内创建的一个探索性图形，并添加有信息量的标题，以便其他人更容易理解。

### 11.3 注释

除了标记图形的主要组件之外，标记单个观测或观测组通常也很有用。你可以使用的第一个工具是 `geom_text()`。`geom_text()` 类似于 `geom_point()`，但它有一个额外的美学属性：`label`。这使得在你的图形中添加文本标签成为可能。

标签的来源有两种可能。首先，你可能有一个提供标签的 tibble。在下面的图形中，我们提取了每种驱动类型中发动机尺寸最大的汽车，并将它们的信息保存为一个名为 `label_info` 的新数据框。

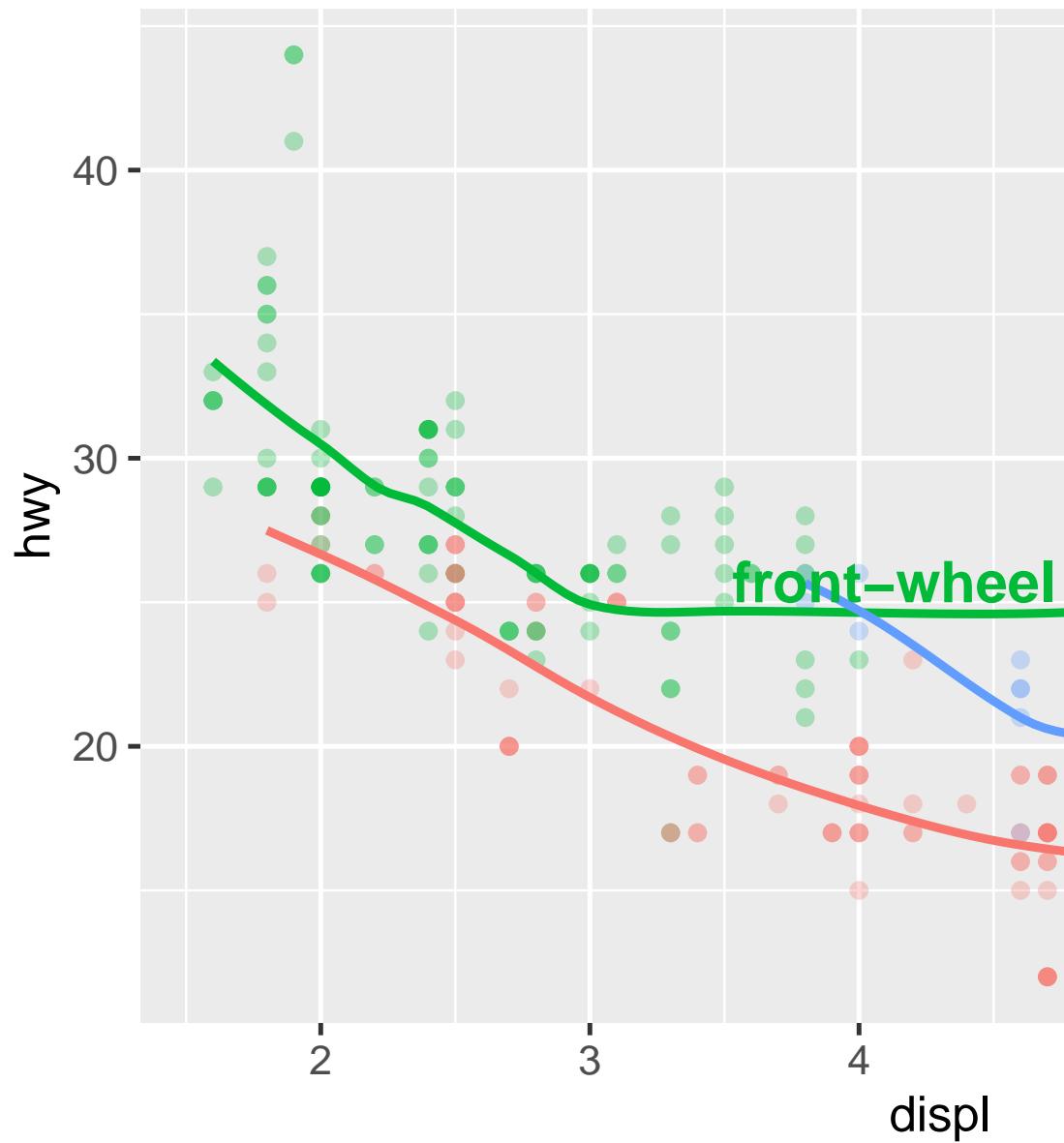
```
label_info <- mpg |>
  group_by(drv) |>
  arrange(desc(displ)) |>
  slice_head(n = 1) |>
  mutate(
    drive_type = case_when(
      drv == "f" ~ "front-wheel drive",
      drv == "r" ~ "rear-wheel drive",
      drv == "4" ~ "4-wheel drive"
    )
  ) |>
  select(displ, hwy, drv, drive_type)
```

```
label_info
#> # A tibble: 3 x 4
#> # Groups:   drv [3]
#>   displ   hwy   drv   drive_type
#>   <dbl> <int> <chr> <chr>
#> 1   6.5     17   4   4-wheel drive
#> 2   5.3     25   f   front-wheel drive
#> 3   7       24   r   rear-wheel drive
```

然后，我们使用这个新的数据框直接标记这三个组，用直接放在图形上的标签替换图例。通过使用 `fontface` 和 `size` 参数，我们可以自定义文本标签的外观。它们比图形上其余文本的字体更大且加粗。（`theme(legend.position = "none")` 会关闭所有图例，我们稍后会详细讨论它。）

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point(alpha = 0.3) +
  geom_smooth(se = FALSE) +
  geom_text(
    data = label_info,
    aes(x = displ, y = hwy, label = drive_type),
    fontface = "bold", size = 5, hjust = "right", vjust = "bottom"
  ) +
  theme(legend.position = "none")
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

11 交流



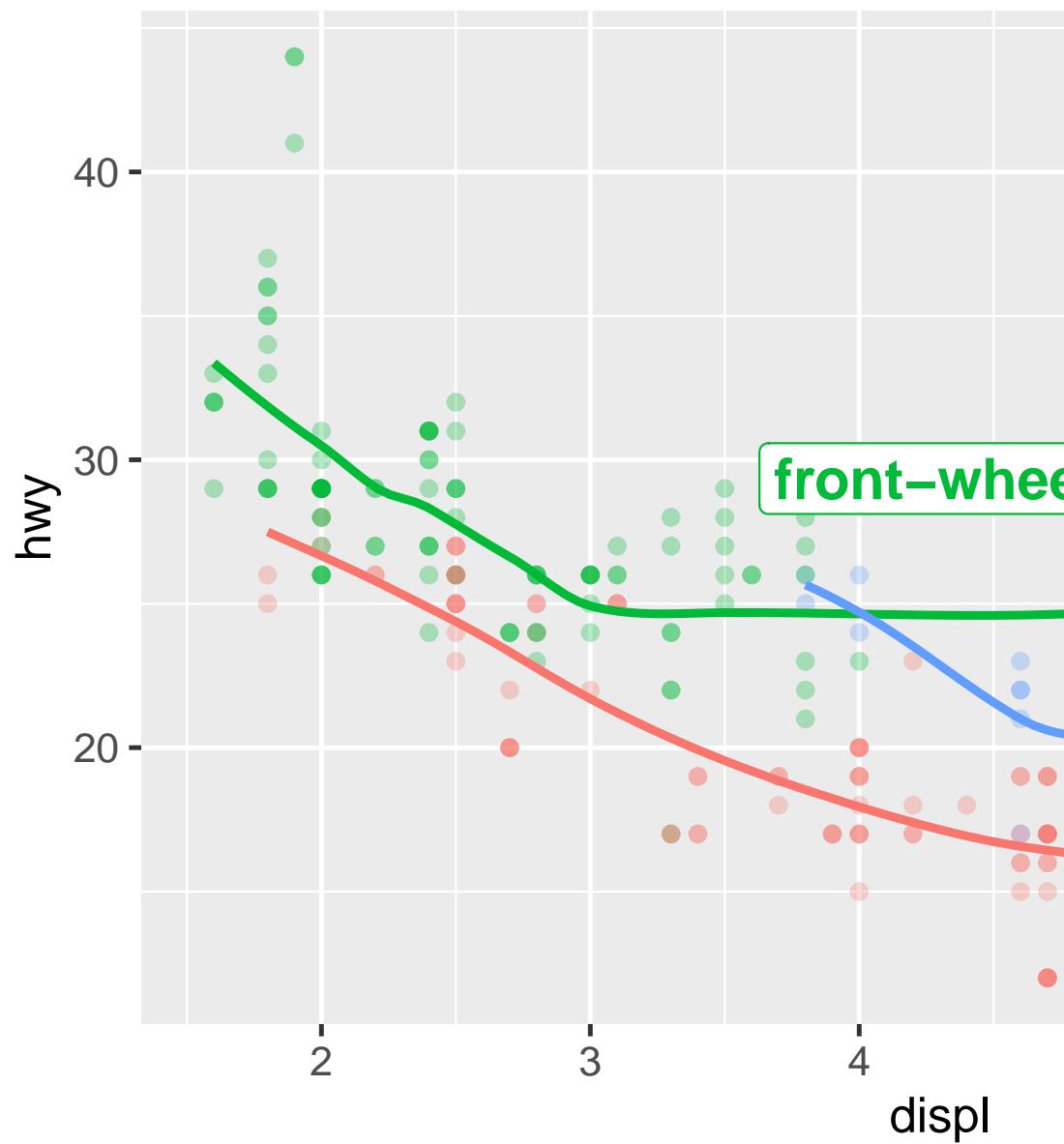
### 11.3 注释

请注意使用 `hjust`（水平对齐）和 `vjust`（垂直对齐）来控制标签的对齐方式。

但是，我们上面制作的带有注释的图形很难阅读，因为标签相互重叠，并且与点重叠。我们可以使用 `ggrepel` 包中的 `geom_label_repel()` 函数来解决这两个问题。这个包会自动调整标签的位置，以避免它们重叠：

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +  
  geom_point(alpha = 0.3) +  
  geom_smooth(se = FALSE) +  
  geom_label_repel(  
    data = label_info,  
    aes(x = displ, y = hwy, label = drive_type),  
    fontface = "bold", size = 5, nudge_y = 2  
  ) +  
  theme(legend.position = "none")  
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

11 交流

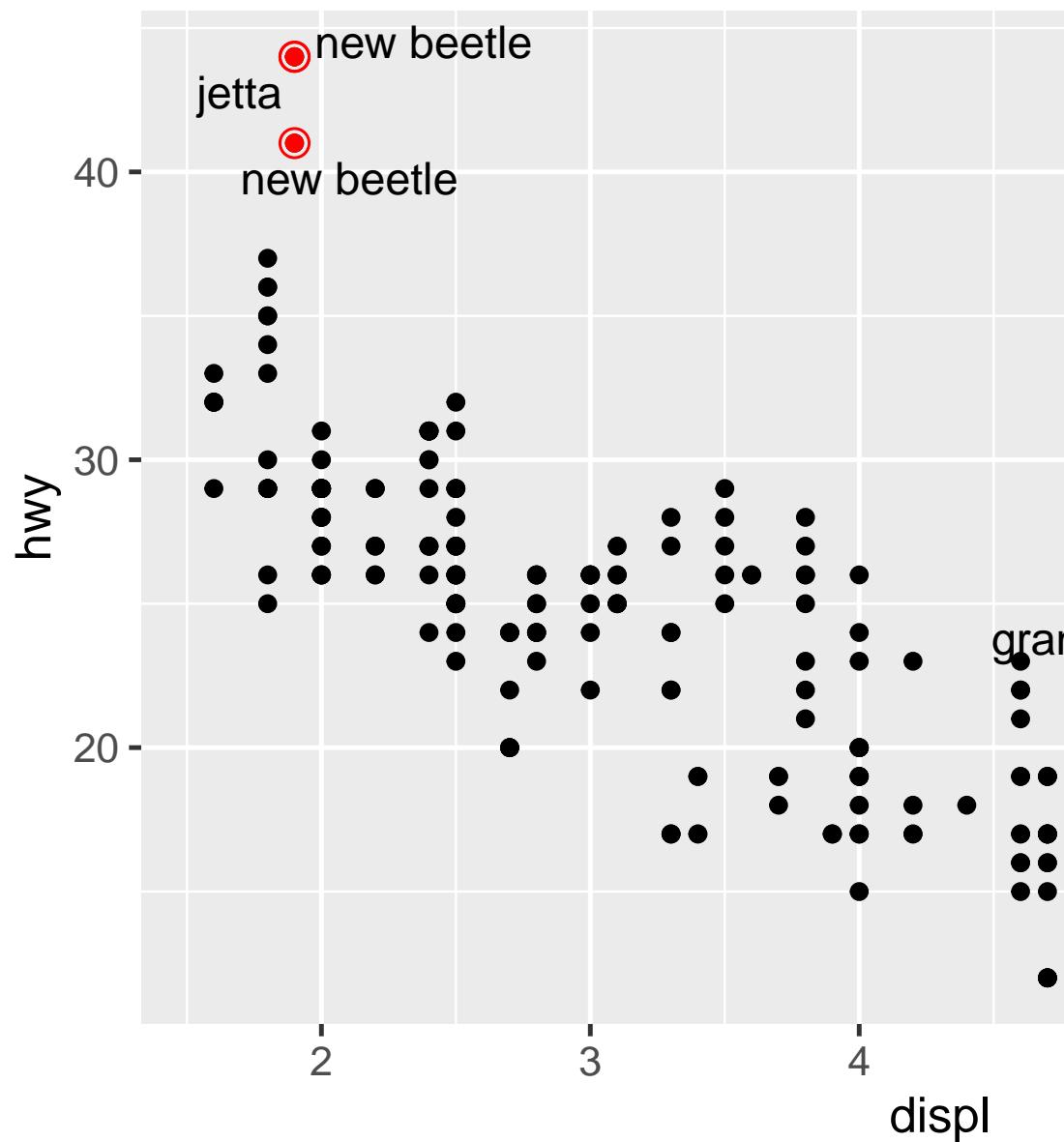


### 11.3 注释

你还可以使用相同的方法来突出显示 `ggrepel` 包中 `geom_text_repel()` 函数在图形上的某些点。请注意这里使用的另一个实用的技巧：我们添加了一层大且空心的点作为第二层，以进一步突出显示带有标签的点。

```
potential_outliers <- mpg |>
  filter(hwy > 40 | (hwy > 20 & displ > 5))

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_text_repel(data = potential_outliers, aes(label = model)) +
  geom_point(data = potential_outliers, color = "red") +
  geom_point(
    data = potential_outliers,
    color = "red", size = 3, shape = "circle open"
  )
```



请记住，除了 `geom_text()` 和 `geom_label()` 之外，`ggplot2` 中还有许多其他 geoms 可用于帮助你注释你的图形。这里有几个想法：

- 使用 `geom_hline()` 和 `geom_vline()` 来添加参考线。我们经常使它们加粗 (`linewidth = 2`) 和为白色 (`color = white`)，并在主要数据层下面绘制它们。这使得它们易于查看，而不会分散对数据的注意力。
- 使用 `geom_rect()` 在感兴趣的点上画一个矩形。矩形的边界由美学属性 `xmin`, `xmax`, `ymin`, `ymax` 定义。另外，请查看 `ggforce` 包，特别是 `geom_mark_hull()`，它允许你使用凸包来注释点的子集。
- 使用带有 `arrow` 参数的 `geom_segment()` 来用箭头吸引对某个点的注意。使用美学属性 `x` 和 `y` 定义起始位置，使用 `xend` 和 `yend` 定义结束位置。

另一个用于向图形添加注释的实用函数是 `annotate()`。作为一般规则，geoms 通常用于突出显示数据的一个子集，而 `annotate()` 则用于向图形添加一个或少数几个注释元素。

为了演示如何使用 `annotate()`，让我们创建一些文本并将其添加到我们的图形中。文本有点长，所以我们将使用 `stringr::str_wrap()` 来根据每行所需的字符数自动添加换行符：

```
trend_text <- "Larger engine sizes tend to have lower fuel economy." |>
  str_wrap(width = 30)
trend_text
#> [1] "Larger engine sizes tend to\nhave lower fuel economy."
```

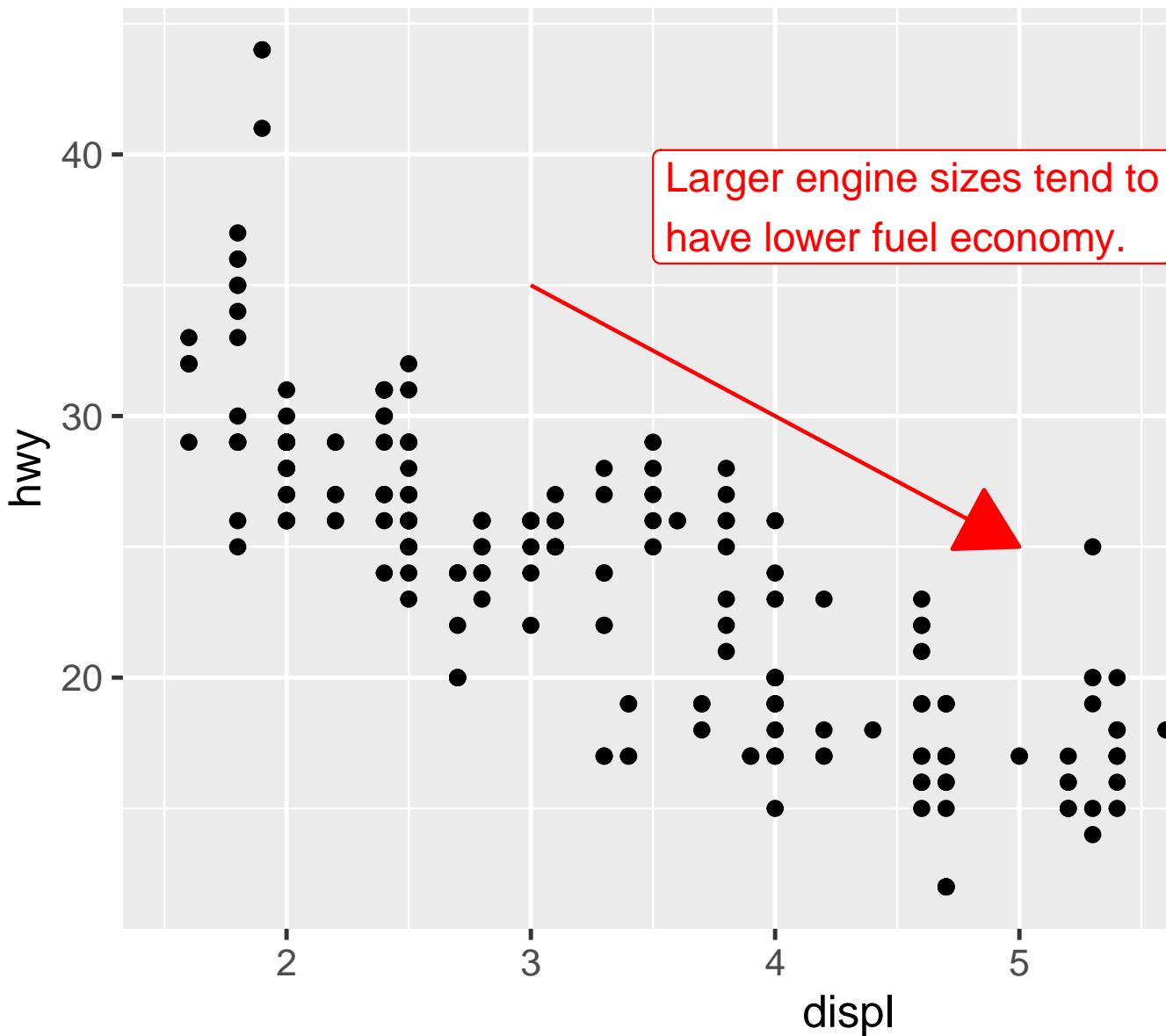
然后，我们添加两层注释：一层使用标签 geom，另一层使用线段 geom。两者中的 `x` 和 `y` 美学属性定义了注释应该从哪里开始，而线段注释中的 `xend`

## 11 交流

和 `yend` 美学属性定义了线段的结束位置。还请注意，线段被设置为箭头的样式。

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  annotate(
    geom = "label", x = 3.5, y = 38,
    label = trend_text,
    hjust = "left", color = "red"
  ) +
  annotate(
    geom = "segment",
    x = 3, y = 35, xend = 5, yend = 25, color = "red",
    arrow = arrow(type = "closed")
  )
```

### 11.3 注释



## 11 交流

注释是一种强大的工具，用于交流你的可视化的主要内容和有趣特性。唯一的限制是你的想象力（以及你耐心地调整注释位置以达到美观的耐心）！

### 11.3.1 练习

1. 使用 `geom_text()` 并通过设置接近边界的坐标值来在图的四个角落放置文本。
2. 使用 `annotate()` 在最后一个图的中心添加一个点，而无需创建一个 `table`。自定义点的形状、大小或颜色。
3. `geom_text()` 中的标签是如何与分面交互的？你如何给单个分面添加一个标签？你如何在每个分面中添加不同的标签？（提示：考虑传递给 `geom_text()` 的数据集）
4. `geom_label()` 的哪些参数控制背景框的外观？
5. `arrow()` 函数的四个参数是什么？它们是如何工作的？创建一系列图表来展示最重要的选项。

## 11.4 比例尺

使图形更易于传达信息的第三种方法是调整比例尺。比例尺控制着美学映射如何以视觉形式展现。

### 11.4.1 默认比例尺

通常，ggplot2 会自动为你添加比例尺。例如，当你输入：

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

ggplot2 在后台自动添加默认比例尺：

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  scale_x_continuous() +
  scale_y_continuous() +
  scale_color_discrete()
```

请注意比例尺的命名方案：以 `scale_` 开头，后面是美学的名称，然后是 `_`，再然后是比例尺的名称。默认的比例尺是根据它们与之对齐的变量类型来命名的：连续、离散、日期时间或日期。`scale_x_continuous()` 将 `displ` 的数值以连续的数字线形式放在 `x` 轴上，`scale_color_discrete()` 为每种车型类别选择颜色，等等。下面将介绍许多非默认的比例尺。

默认的比例尺已经被精心选择，以便为广泛的输入提供良好的工作效果。然而，你可能想要出于两个原因覆盖默认值：

- 你可能想要调整默认比例尺的一些参数。这允许你做一些事情，比如改变轴上的刻度或图例上的键标签。
- 你可能想要完全替换比例尺，并使用完全不同的算法。通常，由于你更了解数据，所以你可以做得比默认值更好。

## 11 交流

### 轴刻度和图例键

轴和图例统称为“指南”。轴用于 x 和 y 美学；图例用于其他所有内容。

有两个主要参数影响轴上的刻度和图例上的键的外观：`breaks` 和 `labels`。  
`breaks` 控制刻度的位置或与键关联的值。`labels` 控制与每个刻度/键关联的文本标签。`breaks` 的最常见用法是覆盖默认选择：

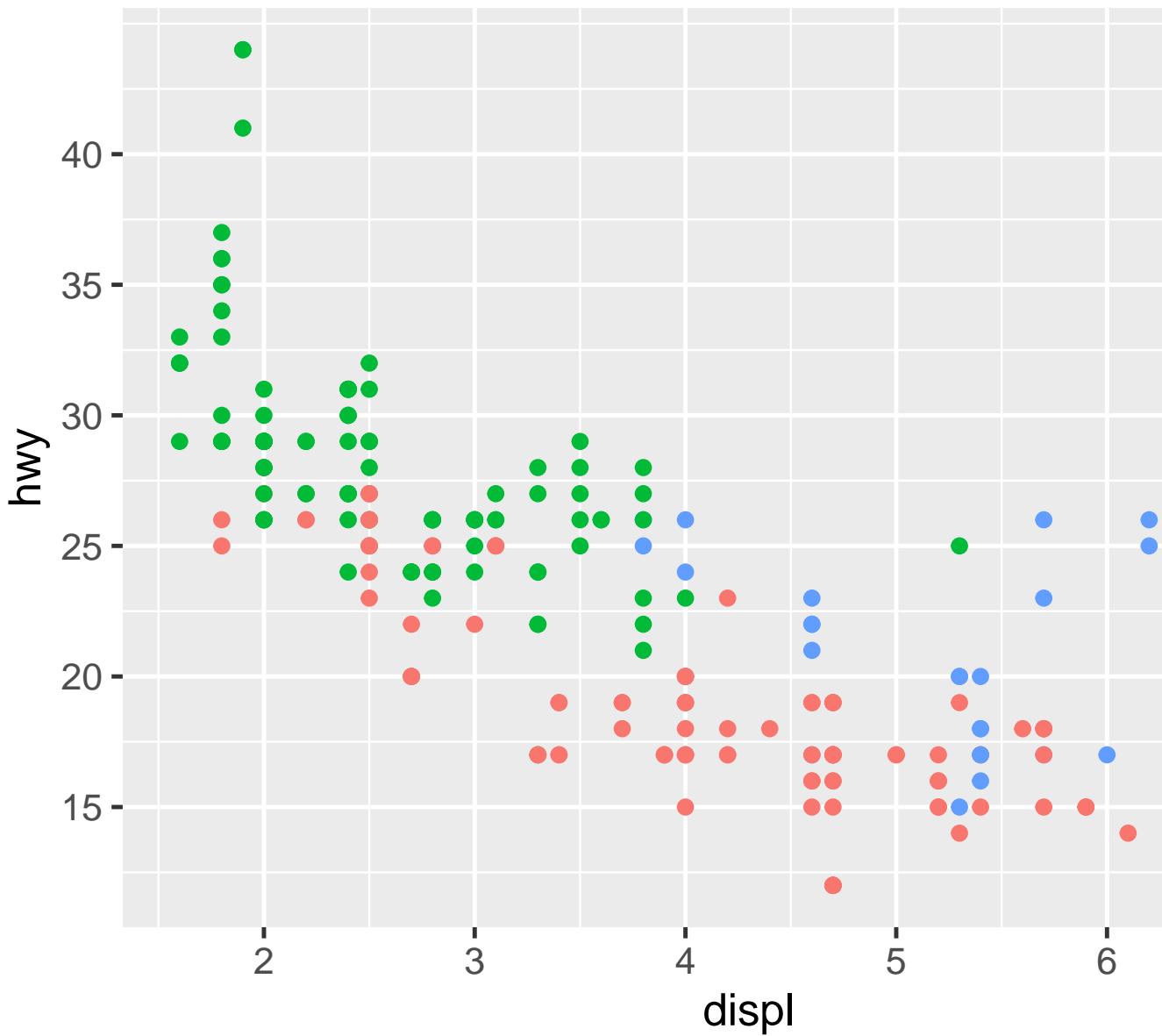
#### 11.4.2 轴刻度和图例键

轴和图例统称为“指南”。轴用于 x 和 y 美学；图例用于其他所有内容。

有两个主要参数影响轴上的刻度和图例上的键的外观：`breaks` 和 `labels`。  
`breaks` 控制刻度的位置或与键关联的值。`labels` 控制与每个刻度/键关联的文本标签。`breaks` 的最常见用法是覆盖默认选择：

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +  
  geom_point() +  
  scale_y_continuous(breaks = seq(15, 40, by = 5))
```

11.4 比例尺



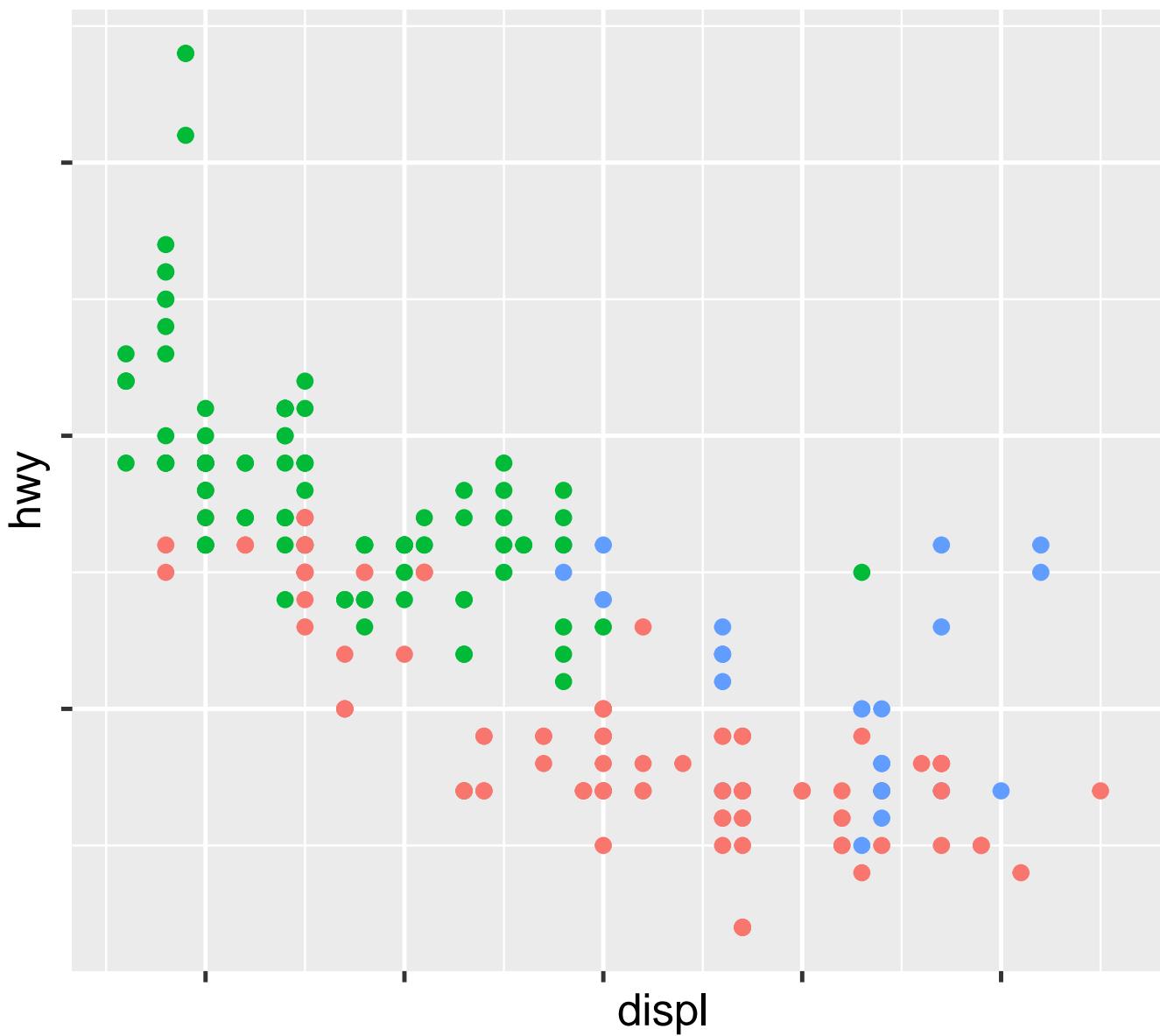
353

## 11 交流

你可以以相同的方式使用 `labels` (一个与 `breaks` 相同长度的字符向量), 但你也可以将其设置为 `NULL` 以完全抑制标签的显示。这在地图或发布图表时非常有用, 因为在这些情况下你可能无法分享绝对数值。你还可以使用 `breaks` 和 `labels` 来控制图例的外观。对于分类变量的离散比例尺, `labels` 可以是一个命名列表, 其中列出了现有水平名称以及为它们指定的标签。

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(labels = NULL) +
  scale_color_discrete(labels = c("4" = "4-wheel", "f" = "front", "r" = "rear"))
```

11.4 比例尺



## 11 交流

`labels` 参数 `scales` 包中的标签函数结合使用，还可用于将数字格式化为货币、百分比等。左边的图形显示了使用 `label_dollar()` 的默认标签，它添加了美元符号和千位分隔符逗号。右边的图形通过进一步定制，将美元值除以 1,000 并添加后缀“K”（代表“千”），以及添加自定义刻度。请注意，`breaks` 是基于数据的原始尺度。

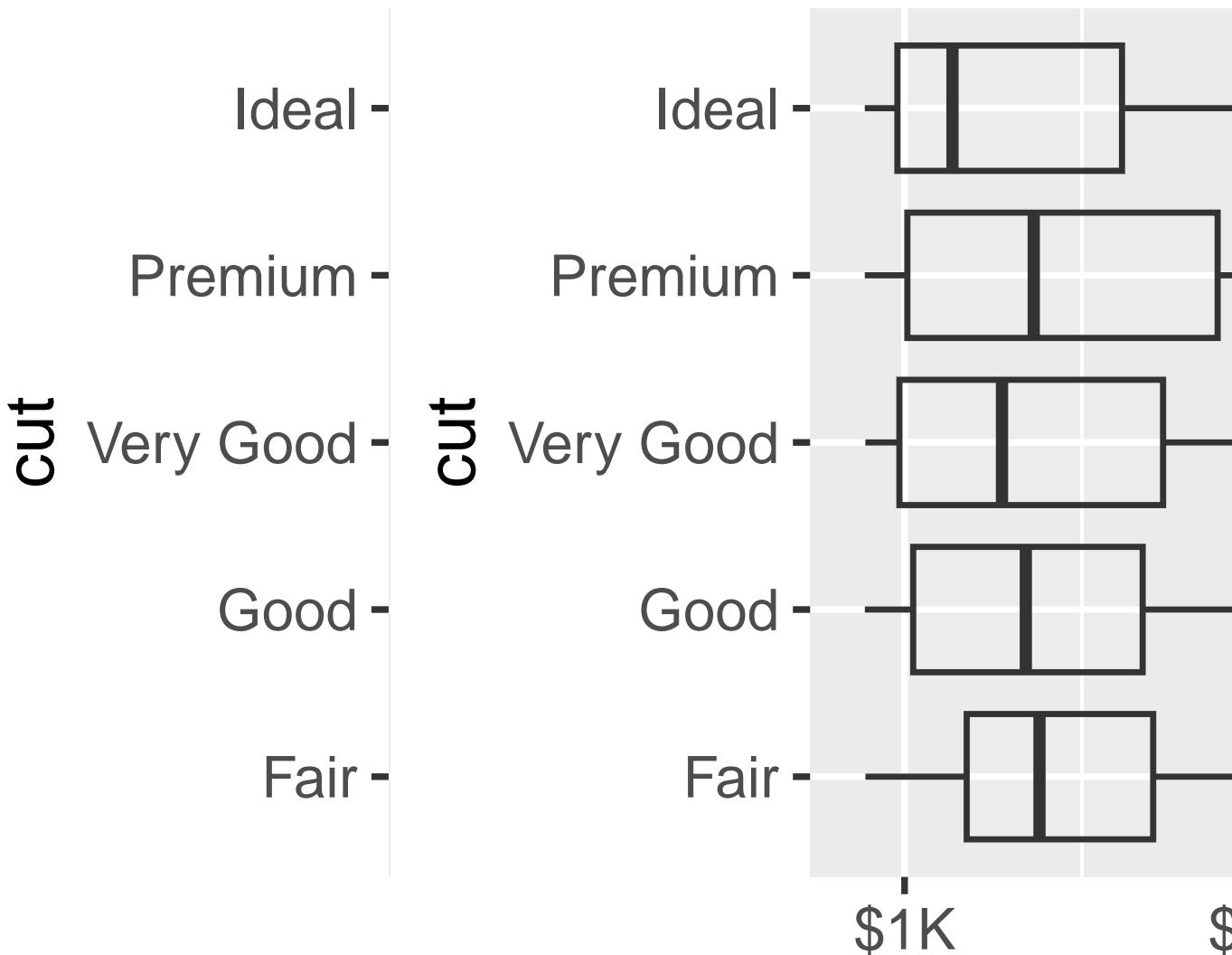
```
# Left
ggplot(diamonds, aes(x = price, y = cut)) +
  geom_boxplot(alpha = 0.05) +
  scale_x_continuous(labels = label_dollar())

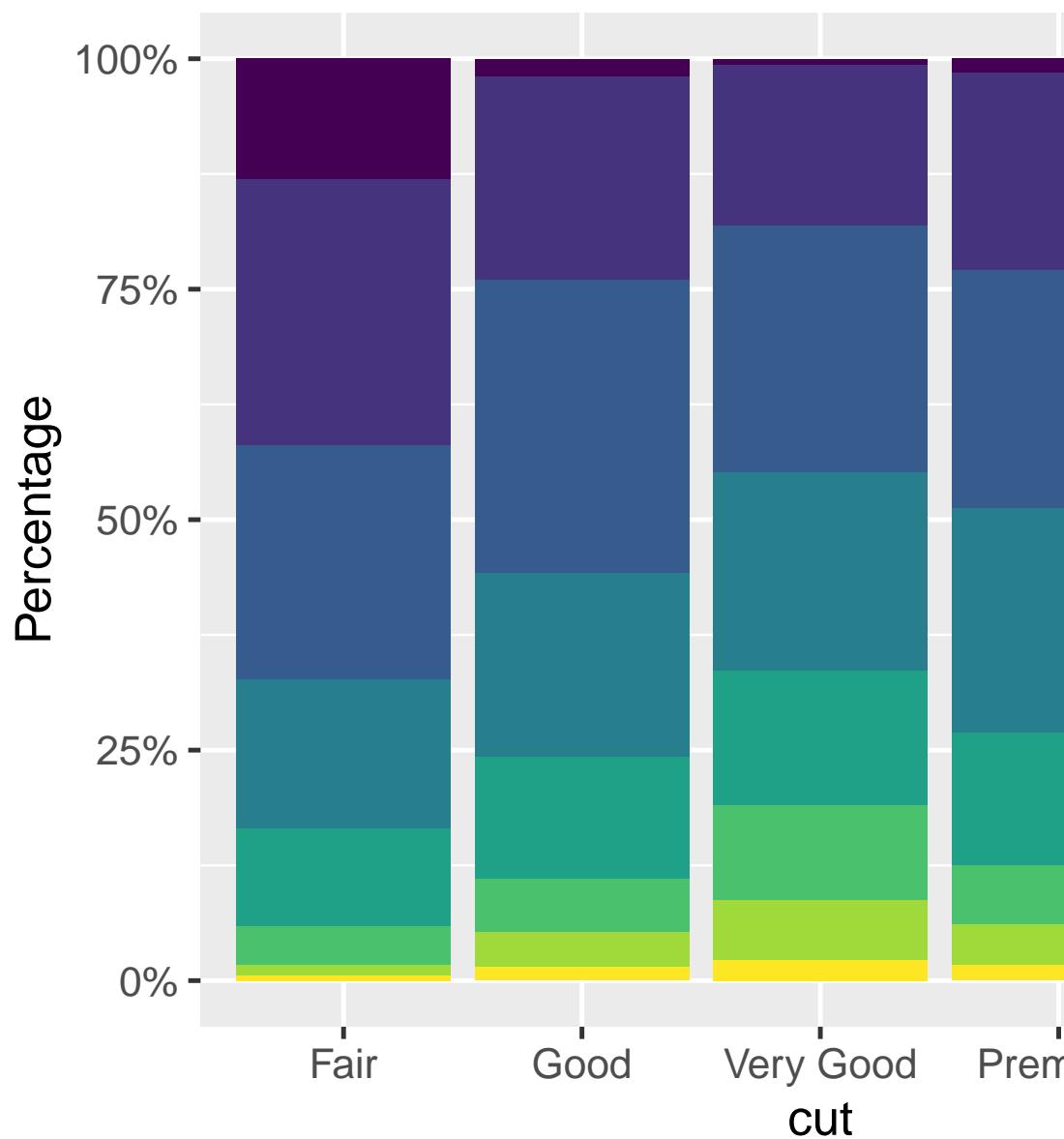
# Right
ggplot(diamonds, aes(x = price, y = cut)) +
  geom_boxplot(alpha = 0.05) +
  scale_x_continuous(
    labels = label_dollar(scale = 1/1000, suffix = "K"),
    breaks = seq(1000, 19000, by = 6000)
  )
```

另一个方便的标签函数是 `label_percent()`:

```
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position = "fill") +
  scale_y_continuous(name = "Percentage", labels = label_percent())
```

11.4 比例尺



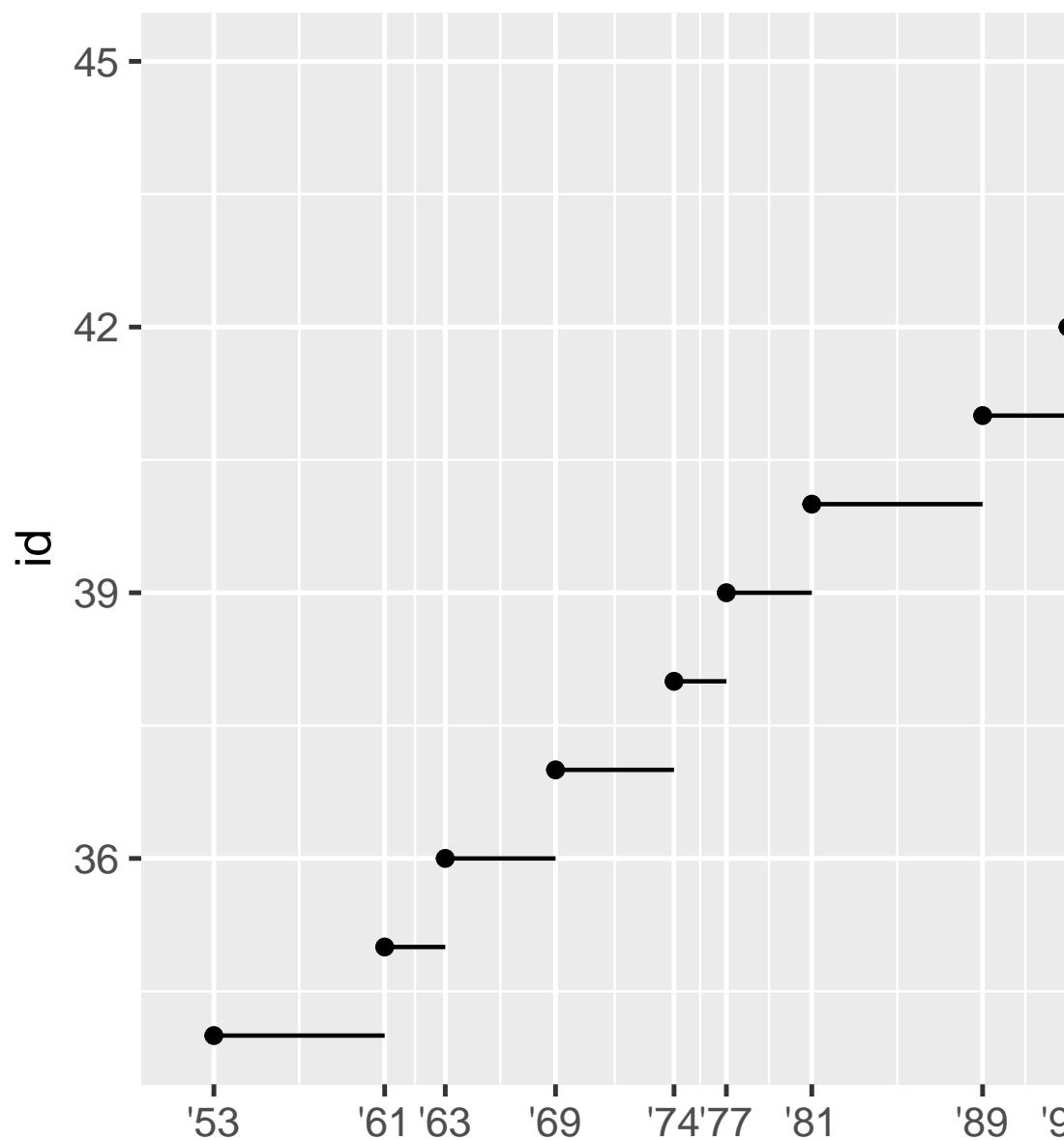


## 11.4 比例尺

`breaks` 的另一个用途是当你的数据点相对较少，并且希望准确地突出显示观测发生的位置时。例如，这张图显示了每位美国总统任期的开始和结束时间。

```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_x_date(name = NULL, breaks = presidential$start, date_labels = "'%y")
```

11 交流



请注意，对于 `breaks` 参数我们提取了 `start` 变量作为一个向量，即 `presidential$start`，因为我们不能为这个参数进行美学映射。另外请注意，对于日期和日期时间比例尺的 `breaks` 和 `labels` 的指定略有不同：

- `date_labels` 需要一个格式规范，其形式与 `parse_datetime()` 相同。
- `date_breaks`（这里未展示）需要一个字符串，如“2 days”或“1 month”。

### 11.4.3 图例布局

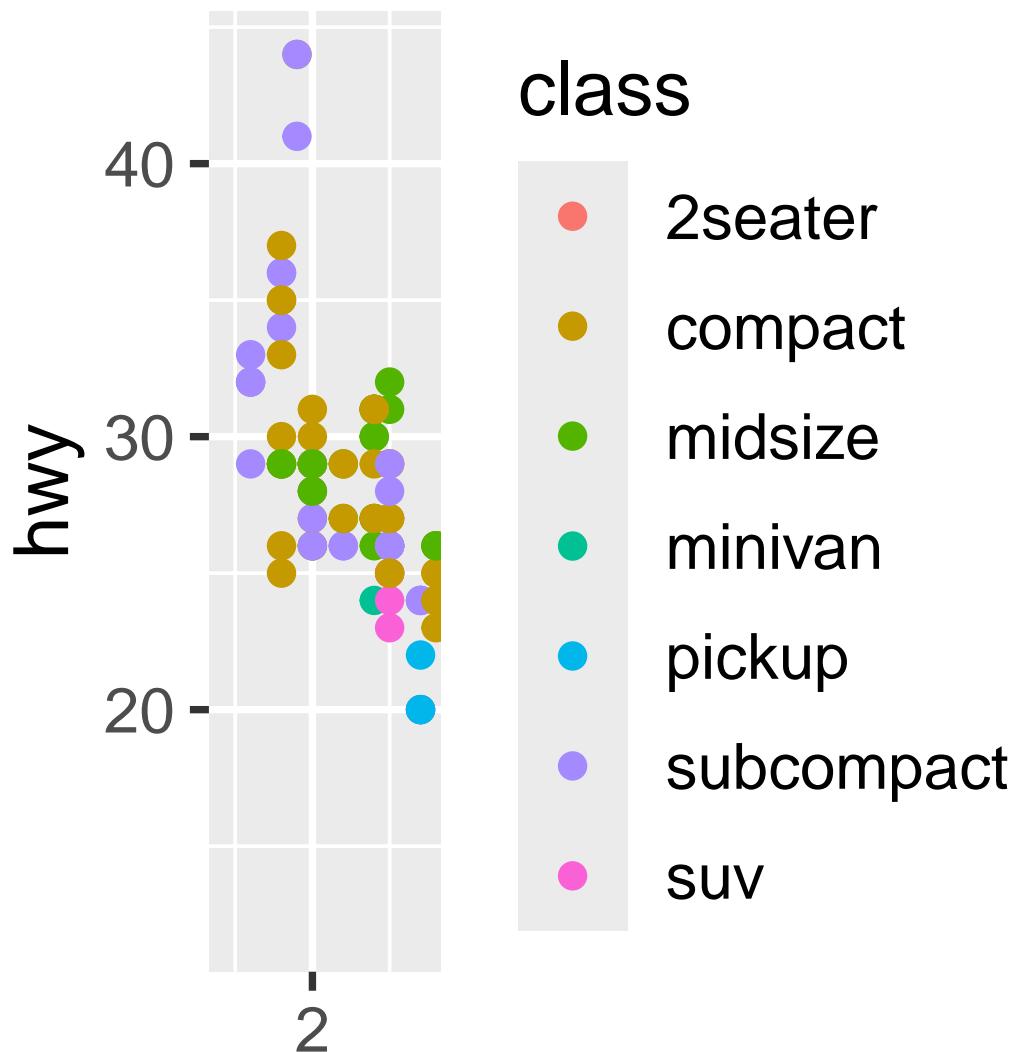
你通常会使用 `breaks` 和 `labels` 来调整坐标轴。虽然它们也适用于图例，但还有一些你更可能使用的其他技术。

要控制图例的整体位置，你需要使用 `theme()` 设置。我们将在本章末尾再次回到主题设置上，但简而言之，它们控制图中非数据部分的内容。主题设置 `legend.position` 控制图例的绘制位置：

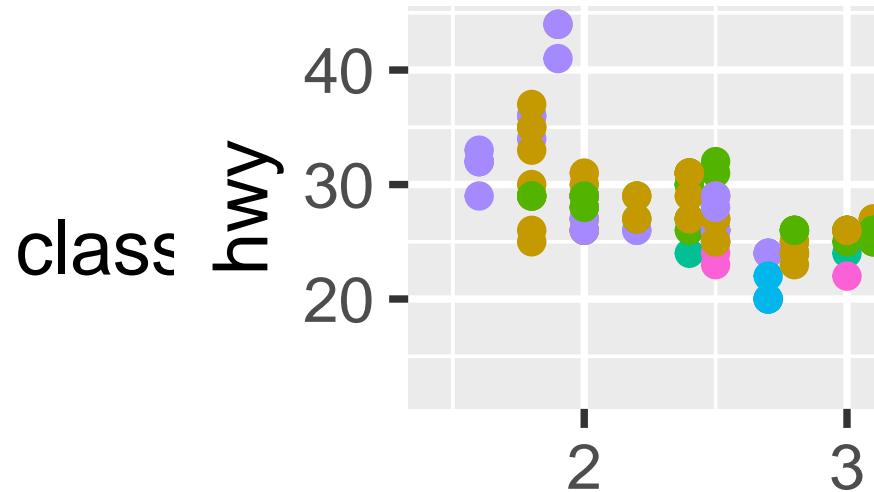
```
base <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class))

base + theme(legend.position = "right") # the default
base + theme(legend.position = "left")
base +
  theme(legend.position = "top") +
  guides(color = guide_legend(nrow = 3))
base +
  theme(legend.position = "bottom") +
  guides(color = guide_legend(nrow = 3))
```

11 交流



362



## 11.4 比例尺

如果你的图是短而宽的，可以将图例放在顶部或底部；如果它是高而窄的，可以将图例放在左侧或右侧。你还可以使用 `legend.position = "none"` 来完全抑制图例的显示。

要控制单个图例的显示，请使用 `guides()` 搭配 `guide_legend()` 或 `guide_colorbar()`。以下示例展示了两个重要的设置：使用 `nrow` 控制图例使用的行数，以及通过覆盖其中一个美学属性来使点变得更大。这在你在图上使用低透明度（`alpha`）来显示许多点时特别有用。

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme(legend.position = "bottom") +
  guides(color = guide_legend(nrow = 2, override.aes = list(size = 4)))
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



注意，`guides()` 中的参数名称与美学名称相匹配，就像 `labs()` 一样。

#### 11.4.4 替换比例尺

除了微调一些细节之外，你还可以完全替换比例尺。有两种类型的比例尺你可能最想要替换：连续位置比例尺和颜色比例尺。幸运的是，其他美学属性的原理都是一样的，所以一旦你掌握了位置和颜色，你就能迅速掌握其他比例尺的替换。

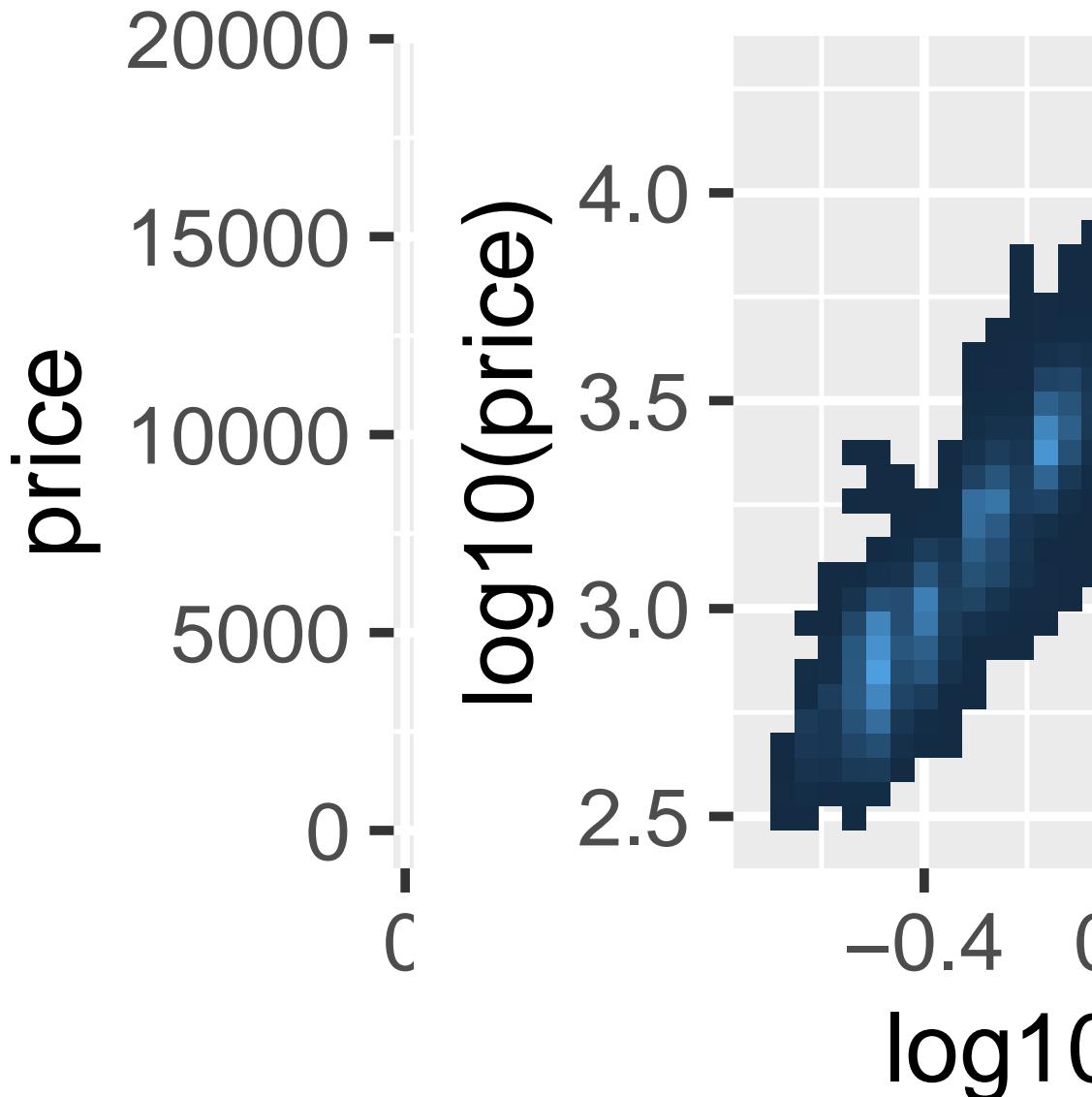
绘制变量的变换是非常有用的。例如，如果我们对钻石的重量（`carat`）和价格（`price`）进行对数变换，就能更容易地看到它们之间的精确关系：

```
# Left
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_bin2d()

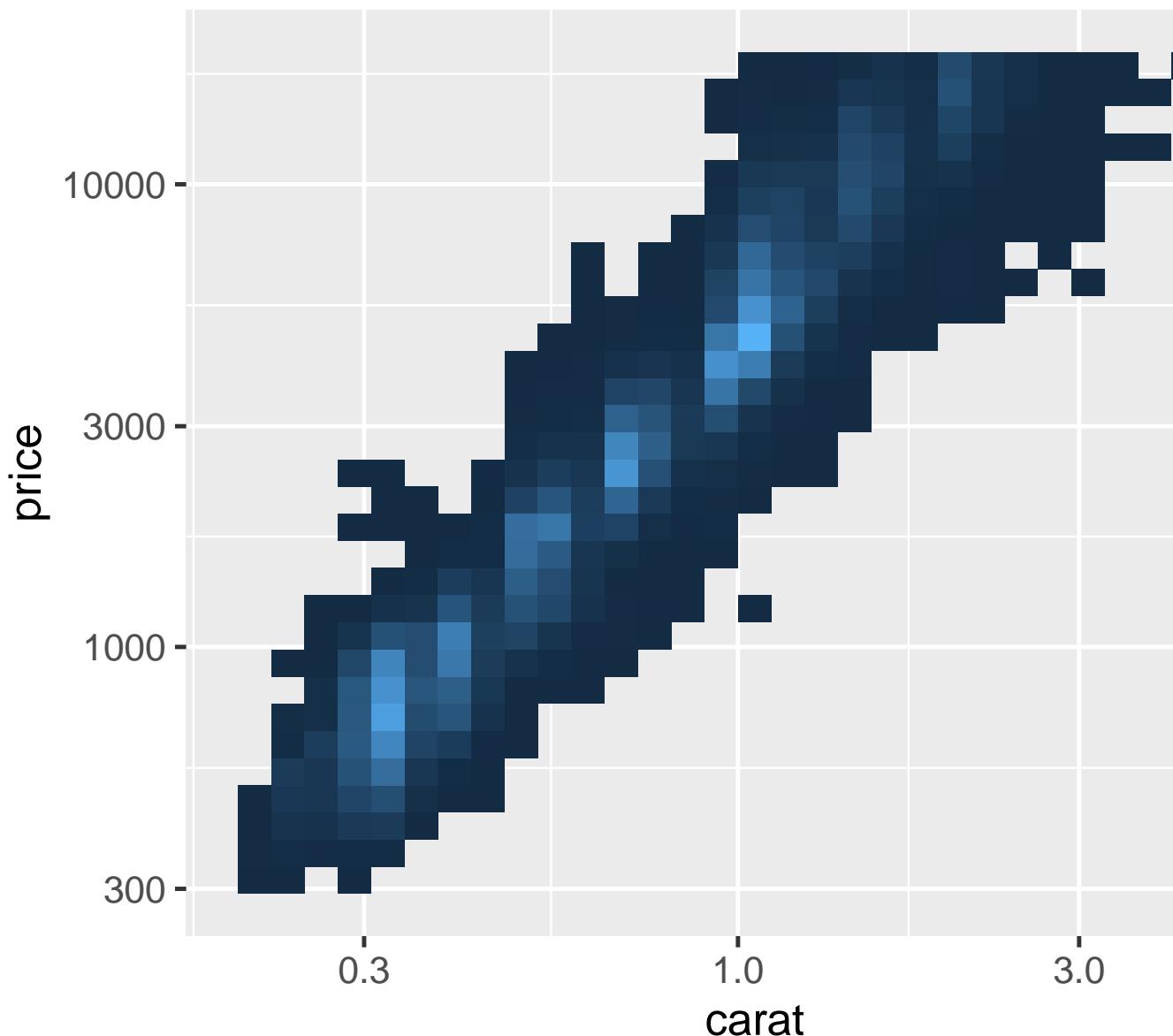
# Right
ggplot(diamonds, aes(x = log10(carat), y = log10(price))) +
  geom_bin2d()
```

然而，这种变换的缺点是坐标轴现在被标记为变换后的值，这使得很难解读该图。与其在美学映射中进行变换，我们可以选择在刻度上进行变换。从视觉上看，两者是相同的，只是坐标轴是按照原始数据的比例尺进行标记的。

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_bin2d() +
  scale_x_log10() +
  scale_y_log10()
```



11.4 比例尺



## 11 交流

另一个经常需要自定义的比例尺是颜色。默认的分类比例尺会选择在色轮上均匀分布的颜色。一个有用的替代方案是 ColorBrewer 比例尺，它经过人工调整以更好地适应患有常见色盲类型的人。下面两个图看起来相似，但红色和绿色的色调差异足够大，以至于右边的点即使对于患有红绿色盲的人也能区分开来<sup>1</sup>。

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(aes(color = drv))  
  
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(aes(color = drv)) +  
  scale_color_brewer(palette = "Set1")
```

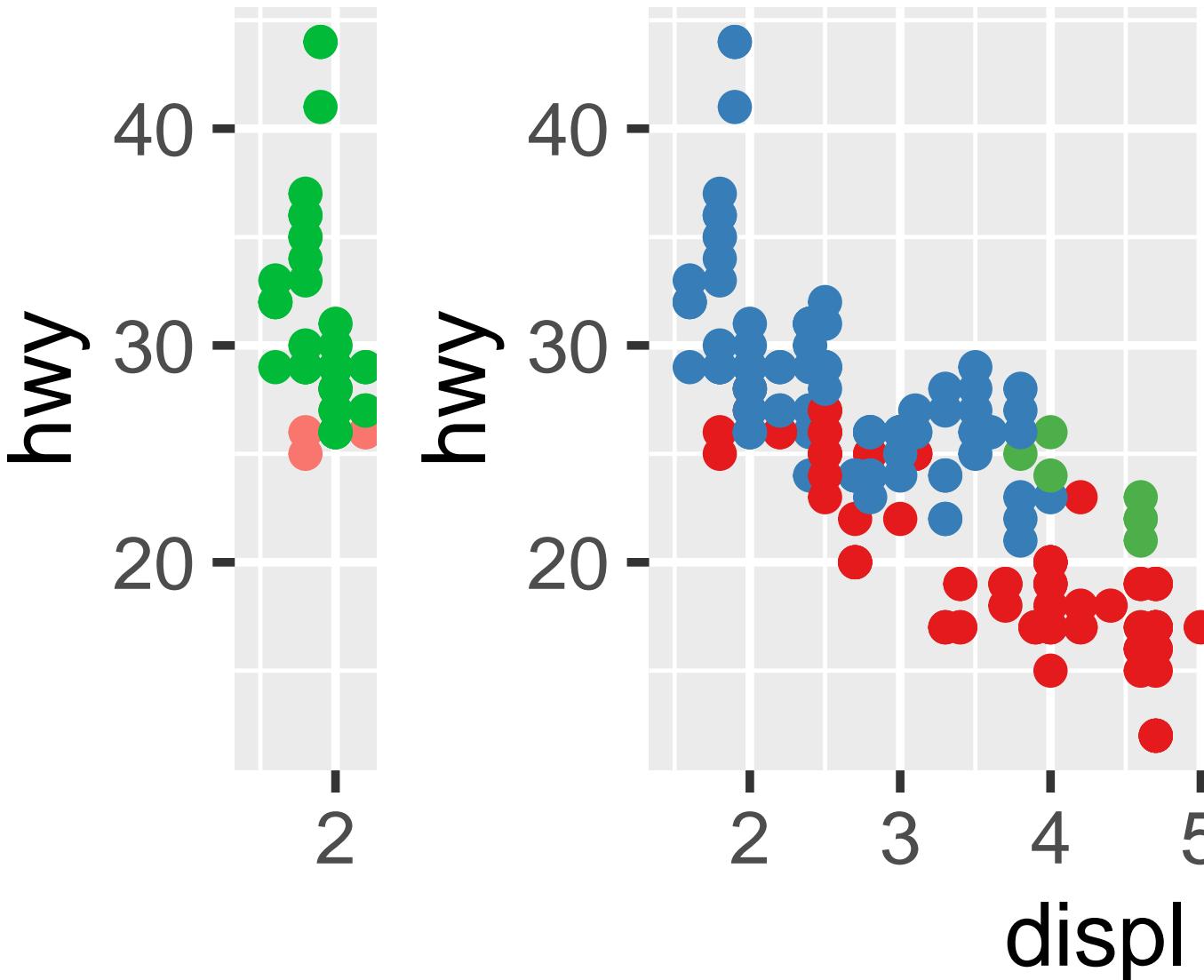
不要忘记提高可访问性的更简单的技术。如果只有几种颜色，你可以添加一个冗余的形状映射。这也有助于确保你的图形在黑白模式下也能被解读。

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(aes(color = drv, shape = drv)) +  
  scale_color_brewer(palette = "Set1")
```

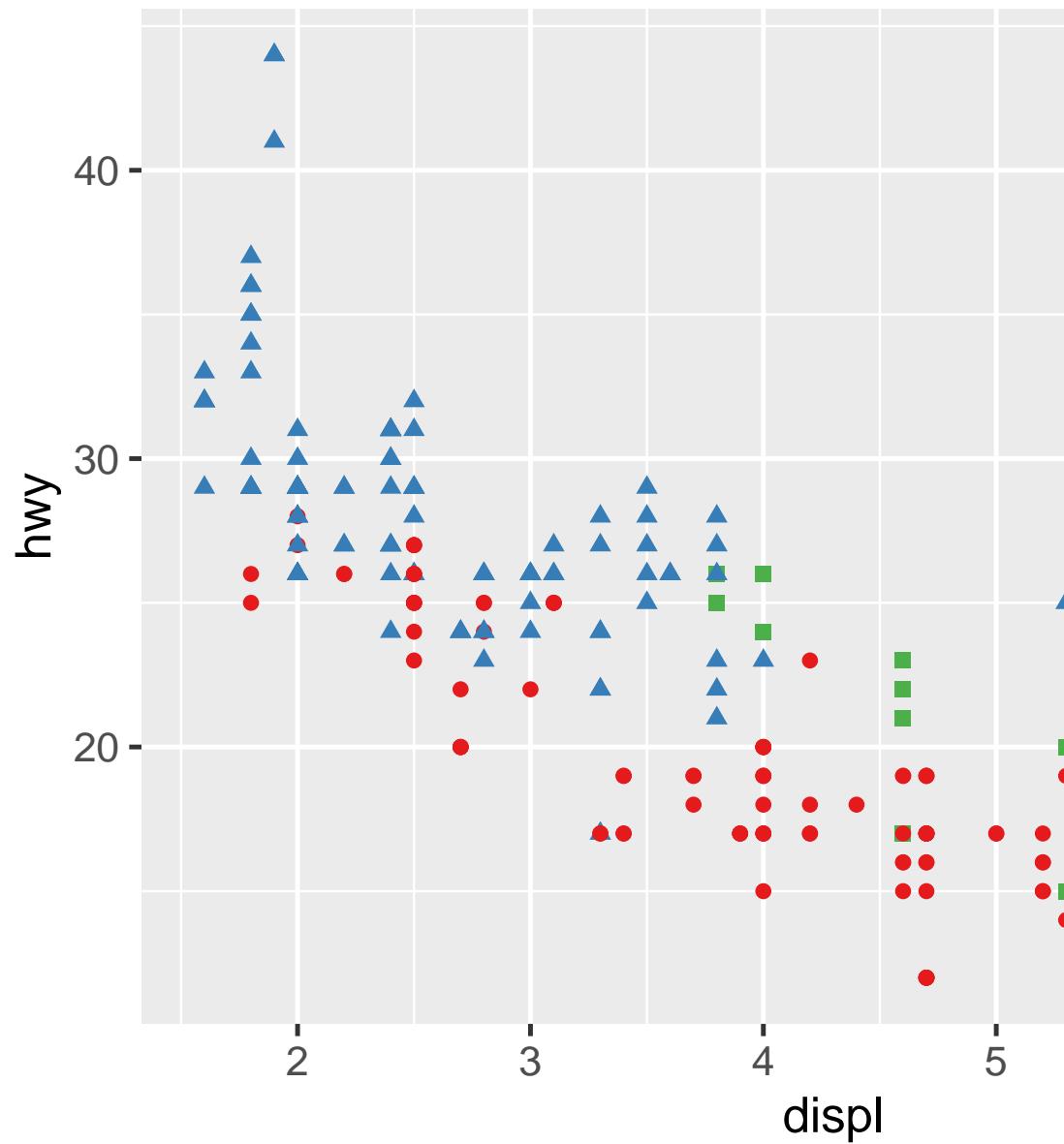
---

<sup>1</sup>您可以使用[SimDaltonism](#)之类的工具来模拟色盲来测试这些图像。

11.4 比例尺



11 交流



## 11.4 比例尺

ColorBrewer 比例尺的文档可以在线查看，网址为<https://colorbrewer2.org/>，并通过 Erich Neuwirth 的 `RColorBrewer` 包在 R 中提供。@ fig-brewer 显示了所有调色板的完整列表。如果你的分类值是有序的，或者有一个“中间值”，那么顺序（顶部）和发散（底部）调色板就特别有用。这通常发生在你使用 `cut()` 函数将连续变量转换为分类变量时。

当你有一个预定义的值与颜色之间的映射时，使用 `scale_color_manual()`。例如，如果我们将总统党派映射到颜色，我们希望使用标准的红色代表共和党，蓝色代表民主党。为这些颜色赋值的一种方法是使用十六进制颜色代码：

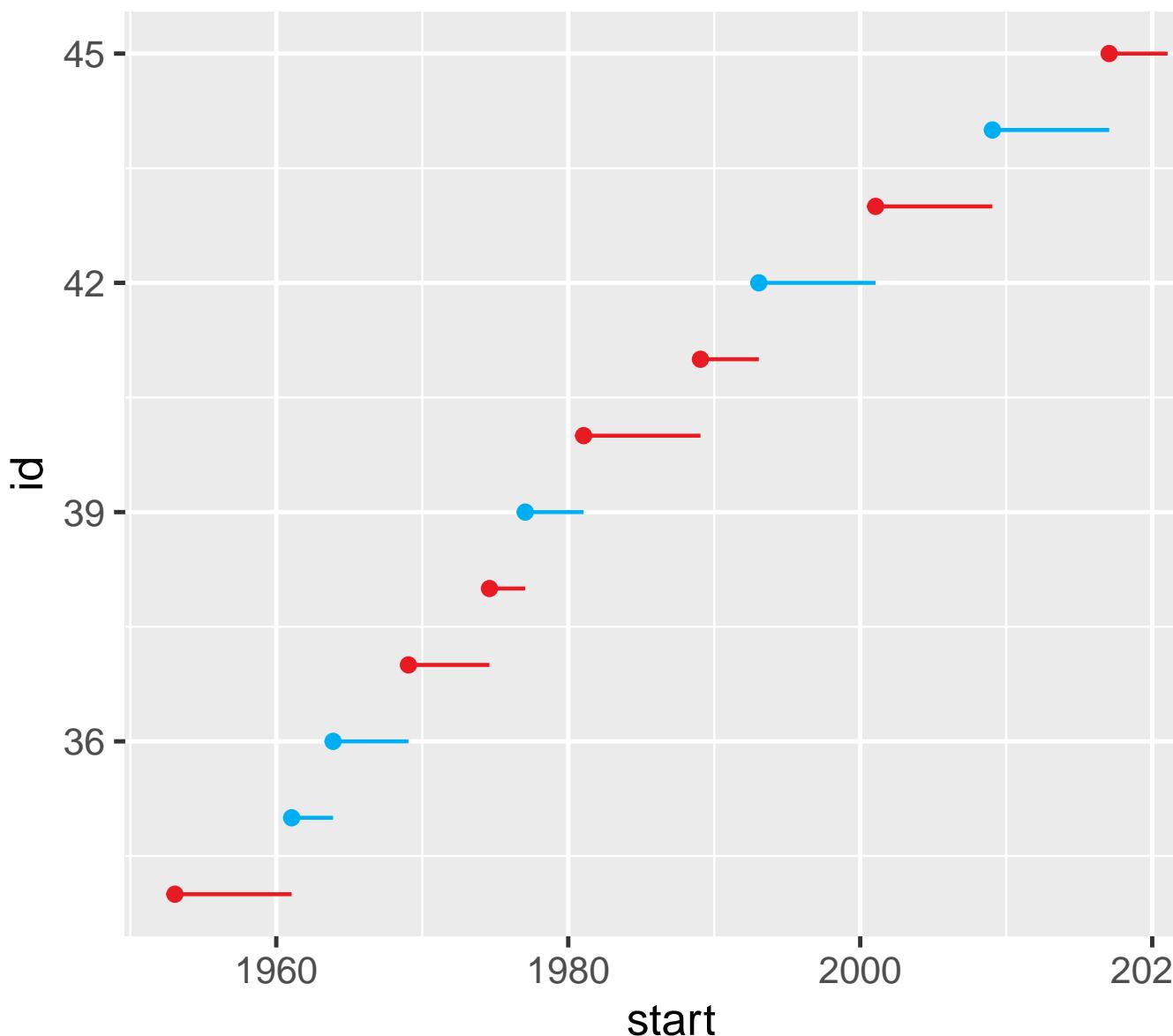
```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(values = c(Republican = "#E81B23", Democratic = "#00AEF3"))
```

## 11 交流



图 11.1: All colorBrewer scales.

11.4 比例尺



## 11 交流

对于连续颜色，可以使用内置的 `scale_color_gradient()` 或 `scale_fill_gradient()`。如果有一个发散的比例尺，你可以使用 `scale_color_gradient2()`。这允许你可以给正数和负数分配不同的颜色。有时，当你想区分高于或低于平均值的点时，这也很有用。

另一个选项是使用 viridis 颜色比例尺。其设计者 Nathaniel Smith 和 Stéfan van der Walt 精心制作了连续的颜色方案，这些方案对于患有各种形式色盲的人以及在彩色和黑白模式下都是感知均匀的。这些比例尺在 `ggplot2` 中作为连续 (c)、离散 (d) 和分箱 (b) 调色板提供。

```
df <- tibble(
  x = rnorm(10000),
  y = rnorm(10000)
)

ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed() +
  labs(title = "Default, continuous", x = NULL, y = NULL)

ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed() +
  scale_fill_viridis_c() +
  labs(title = "Viridis, continuous", x = NULL, y = NULL)

ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed()
```

```
scale_fill_viridis_b() +
  labs(title = "Viridis, binned", x = NULL, y = NULL)
```

请注意，所有的颜色比例尺都有两种类型: `scale_color_*`() 和 `scale_fill_*`()，分别用于颜色和填充的美学 (color scales 在英国和美国的拼写中都是可用的)。

### 11.4.5 缩放

控制图形的界限有三种方法：

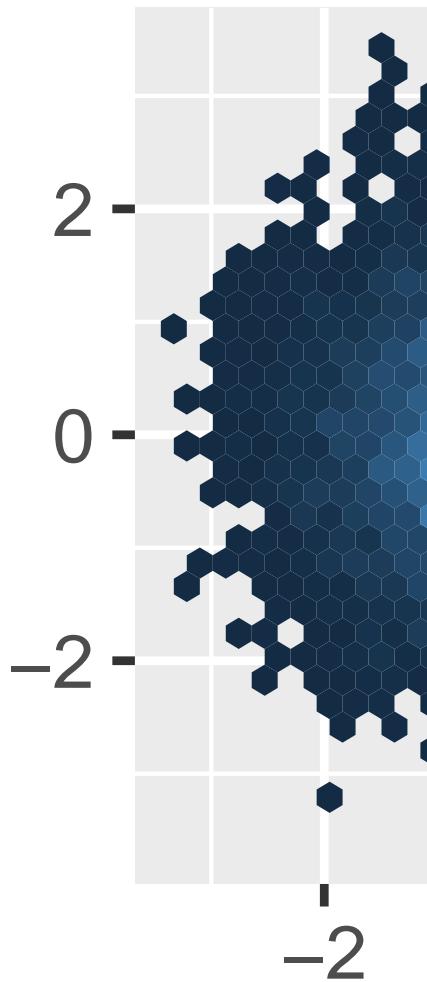
1. 调整要绘制的数据。
2. 在每个比例尺中设置界限。
3. 在 `coord_cartesian()` 中设置 `xlim` 和 `ylim`。

我们将在一系列图形中演示这些选项。左侧的图形显示了发动机大小和燃油效率之间的关系，并按驱动类型着色。右侧的图表显示了相同的变量，但仅绘制了部分数据。子集数据已经影响了 x 和 y 比例尺以及平滑曲线。

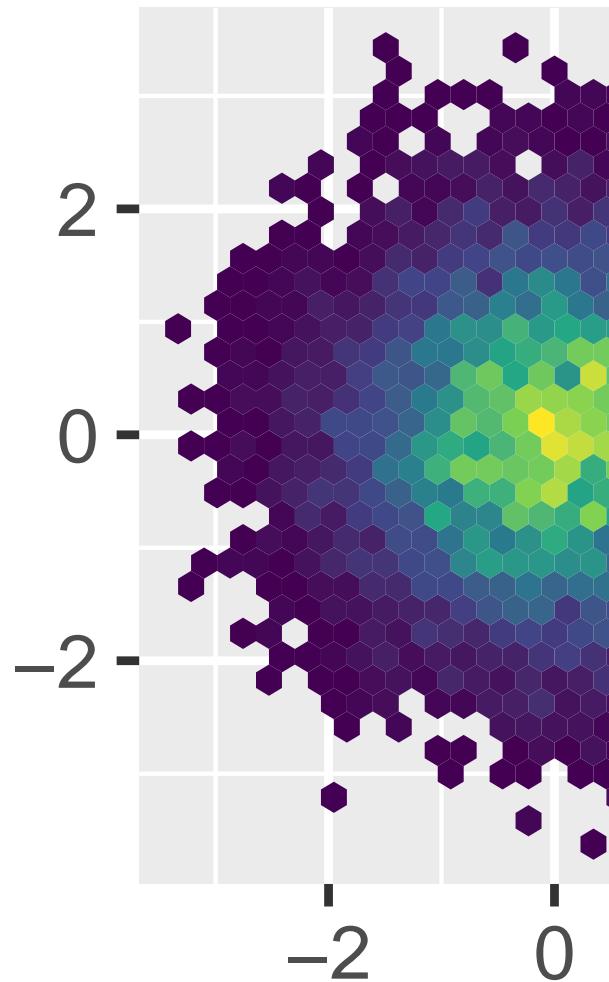
```
# Left
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth()

# Right
mpg |>
  filter(displ >= 5 & displ <= 6 & hwy >= 10 & hwy <= 25) |>
  ggplot(aes(x = displ, y = hwy)) +
```

# Defaul

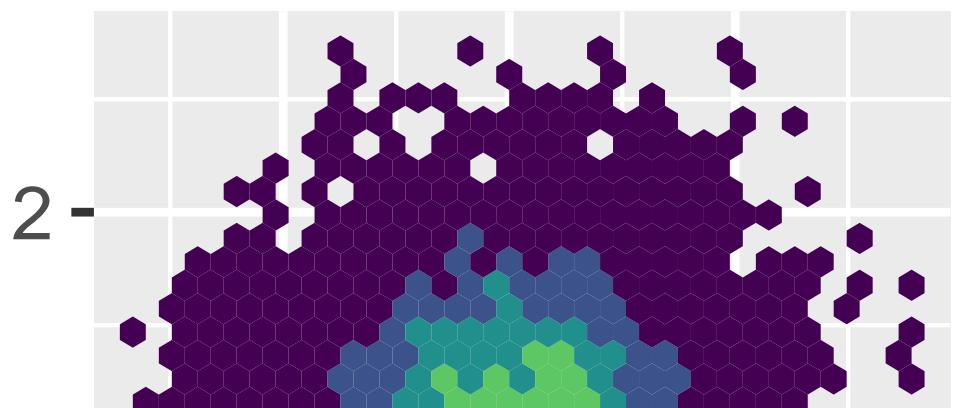


# Viridis, co



# Viridis, binned

376



```
geom_point(aes(color = drv)) +
geom_smooth()
```

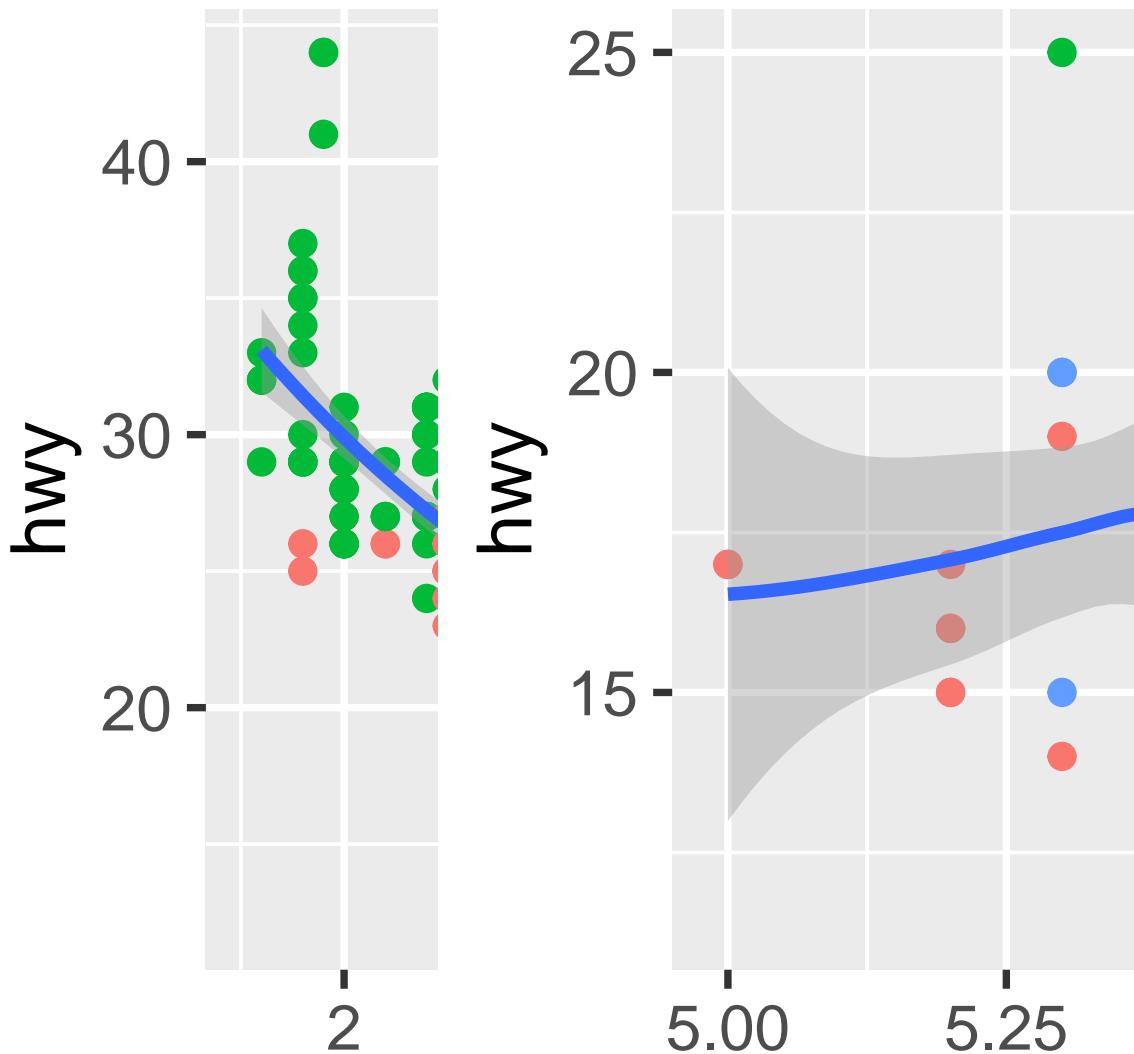
让我们比较下面两个图形，其中左侧的图形在单个比例尺上设置了界限，而右侧的图形在 `coord_cartesian()` 中设置了界限。我们可以看到，减少界限的效果相当于对数据进行了子集选择。因此，为了放大图中的某个区域，通常最好使用 `coord_cartesian()`。

```
# Left
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth() +
  scale_x_continuous(limits = c(5, 6)) +
  scale_y_continuous(limits = c(10, 25))

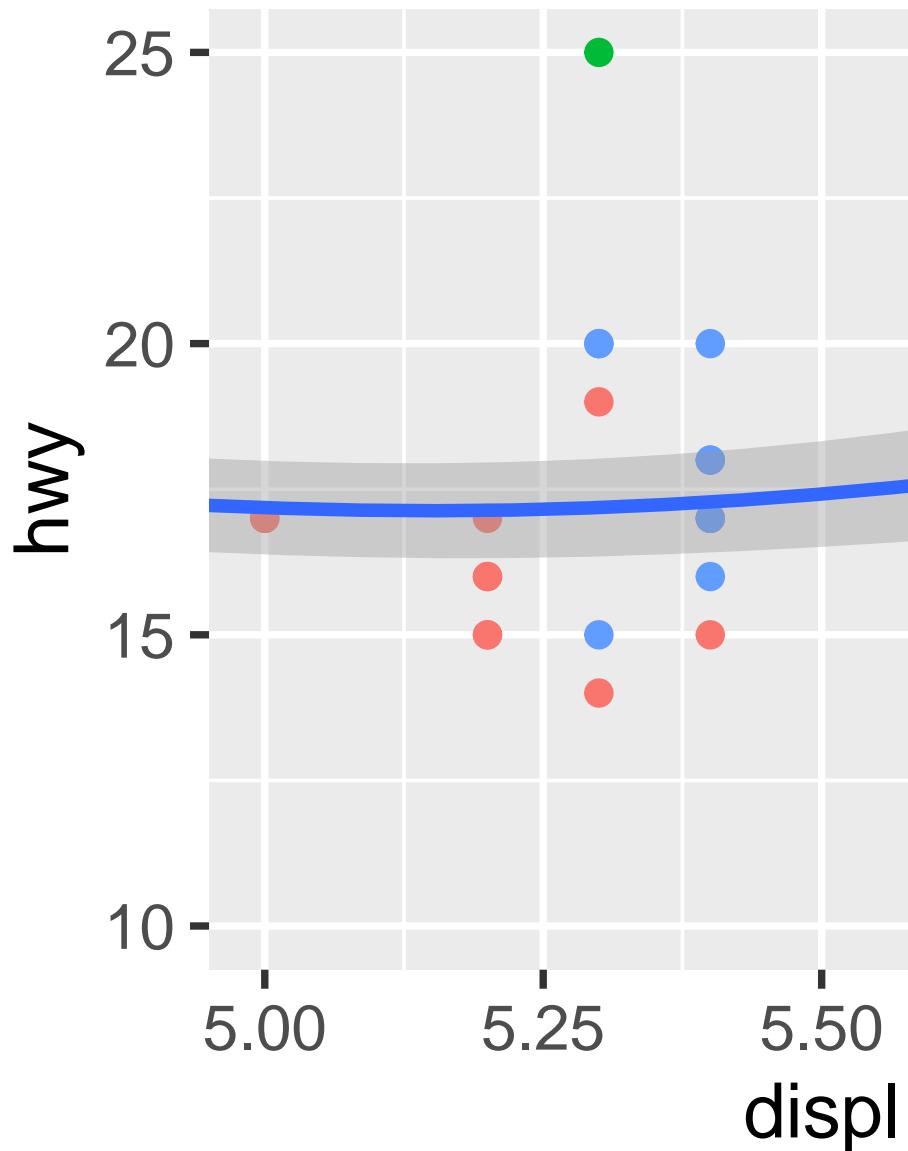
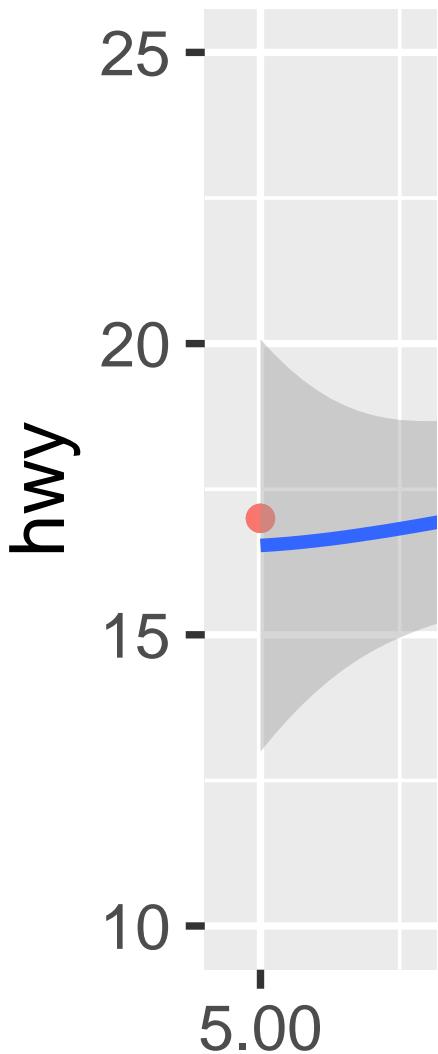
# Right
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth() +
  coord_cartesian(xlim = c(5, 6), ylim = c(10, 25))
```

另一方面，如果你想扩展界限，比如在不同的图形之间匹配比例尺，那么在单个比例尺上设置 `limits` 通常更有用。例如，如果我们提取两类汽车并分别绘制它们，则很难比较这些图形，因为三个比例尺（x 轴、y 轴和颜色美学）的范围都不相同。

11 交流



11.4 比例尺



## 11 交流

```
suv <- mpg |> filter(class == "suv")
compact <- mpg |> filter(class == "compact")

# Left
ggplot(suv, aes(x = displ, y = hwy, color = drv)) +
  geom_point()

# Right
ggplot(compact, aes(x = displ, y = hwy, color = drv)) +
  geom_point()
```

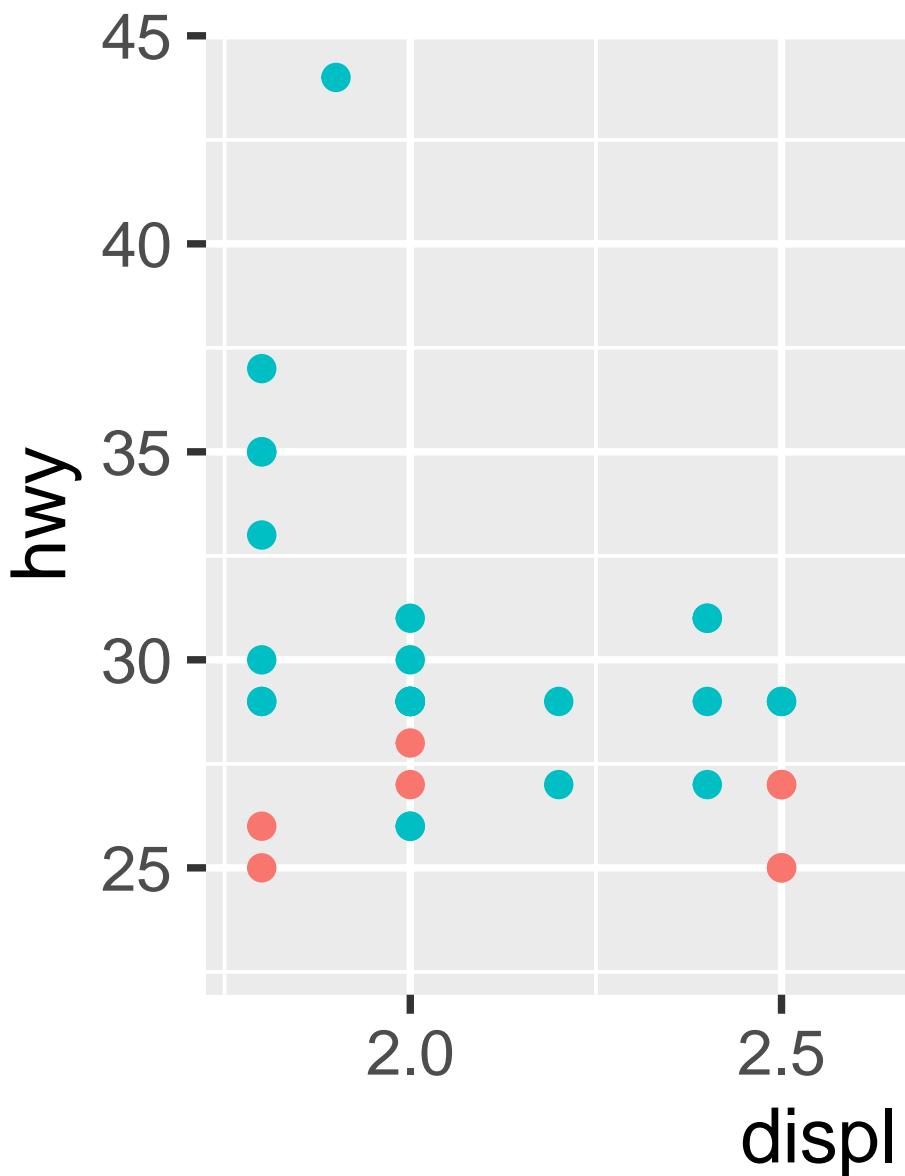
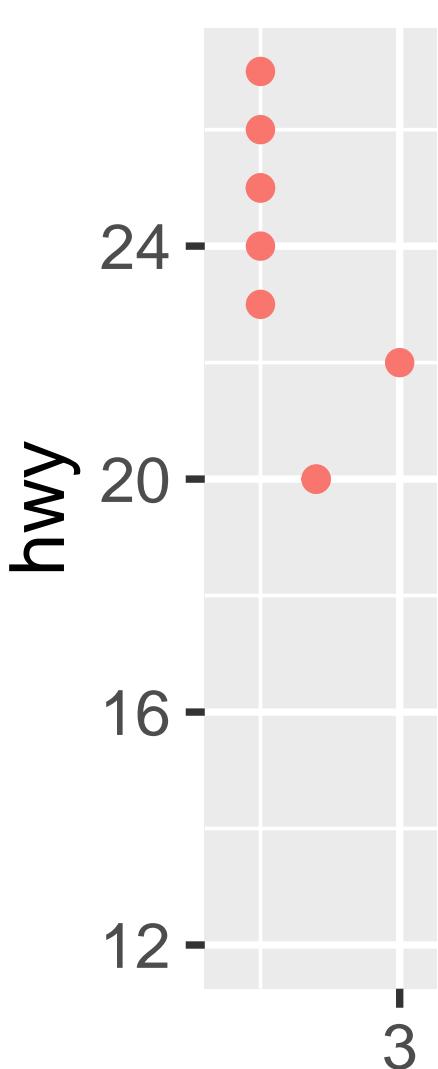
解决这个问题的一种方法是在多个图之间共享比例尺，并使用整个数据的 `limits` 来训练这些比例尺。

```
x_scale <- scale_x_continuous(limits = range(mpg$displ))
y_scale <- scale_y_continuous(limits = range(mpg$hwy))
col_scale <- scale_color_discrete(limits = unique(mpg$drv))

# Left
ggplot(suv, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  x_scale +
  y_scale +
  col_scale

# Right
ggplot(compact, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
```

11.4 比例尺



381

```
x_scale +
y_scale +
col_scale
```

在这种特定情况下，你可以简单地使用分面（faceting）来解决问题，但这种技术更具一般性，例如如果你想在报告的多个页面上分布图形，那么这种技术就很有用。

#### 11.4.6 练习

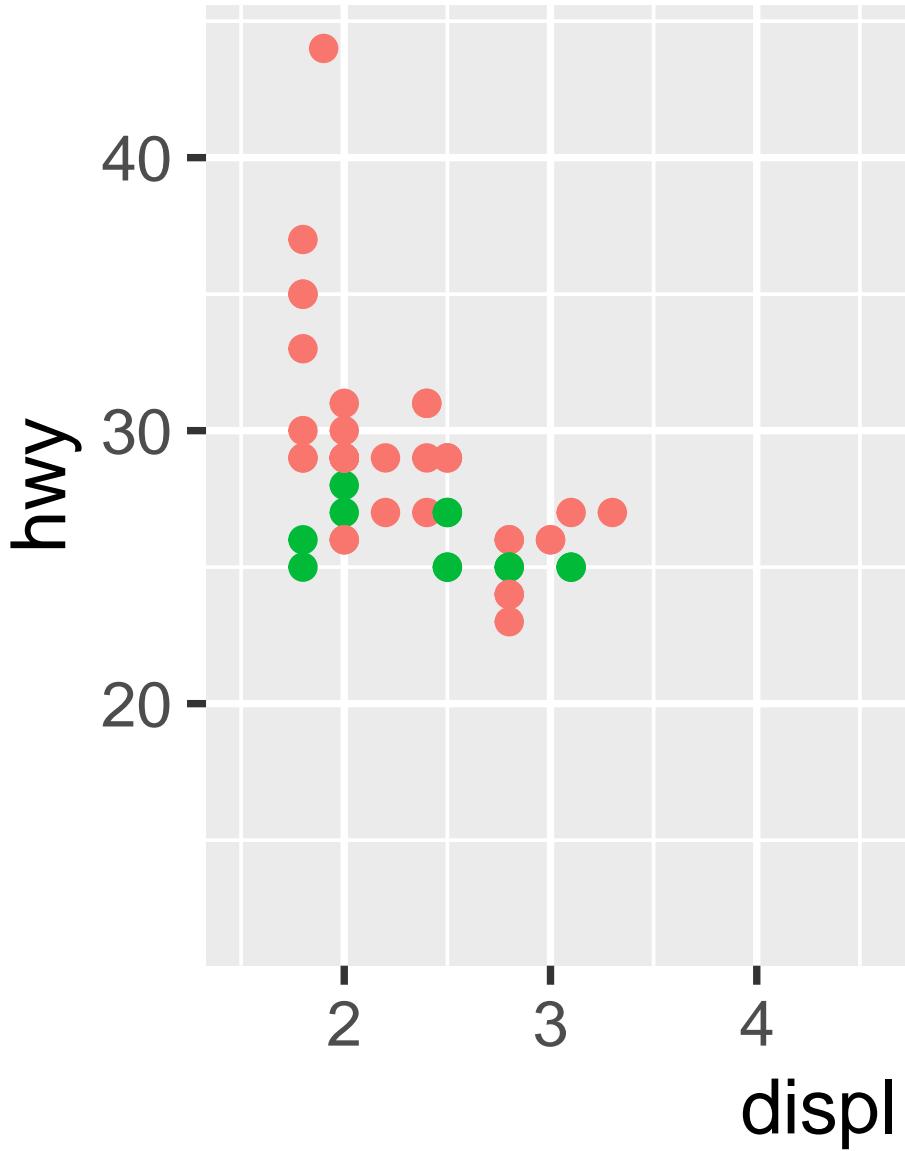
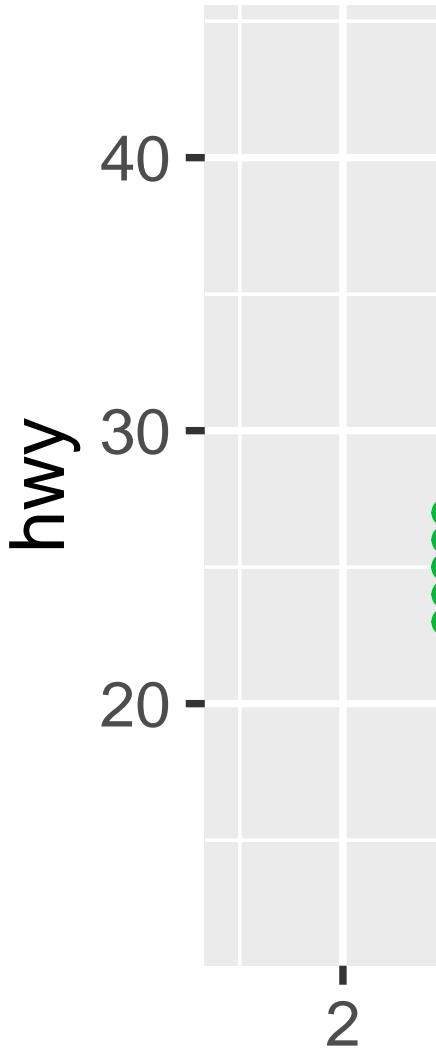
- 为什么下面的代码不覆盖默认的比例尺？

```
df <- tibble(
  x = rnorm(10000),
  y = rnorm(10000)
)

ggplot(df, aes(x, y)) +
  geom_hex() +
  scale_color_gradient(low = "white", high = "red") +
  coord_fixed()
```

- 每个比例尺的第一个参数是什么？它和 `labs()` 相比怎么样？
- 通过以下方式更改总统任期的显示：
  - 结合自定义颜色和 x 轴刻度的两个变体。
  - 改进 y 轴的显示。
  - 为每个任期标上总统的名字。

11.4 比例尺



383

## 11 交流

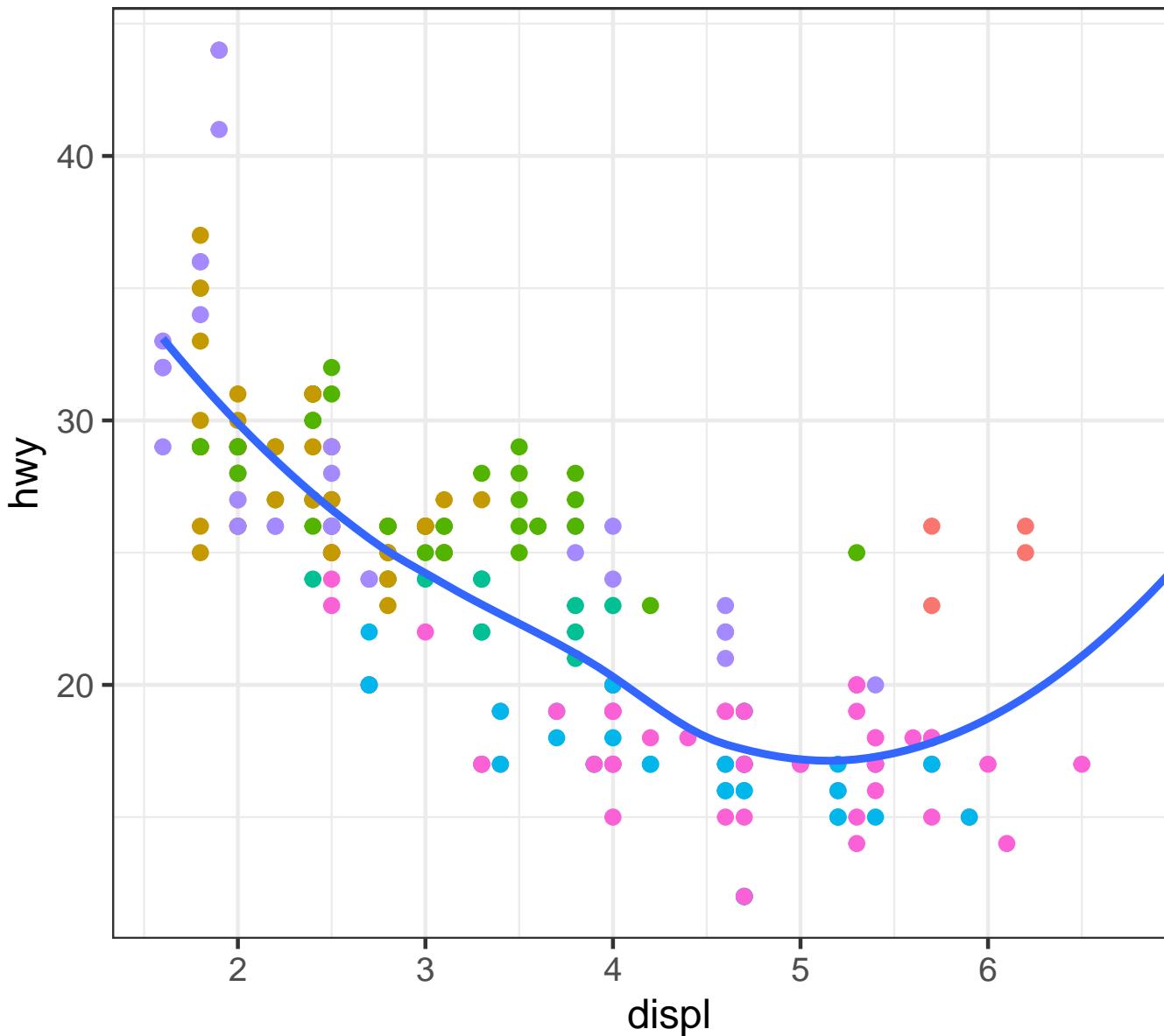
- d. 添加有信息量的图标签。
  - e. 每 4 年设置一个刻度。(这看起来简单但实际上有些棘手!)
4. 首先，创建以下图形。然后，使用 `override.aes` 修改代码以使图例更容易查看。

```
ggplot(diamonds, aes(x = carat, y = price)) +  
  geom_point(aes(color = cut), alpha = 1/20)
```

## 11.5 主题

最后，使用主题来定制图形中的非数据元素：

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(aes(color = class)) +  
  geom_smooth(se = FALSE) +  
  theme_bw()
```



## 11 交流

`ggplot2` 包含了图 ?? 中所示的八种主题，其中 `theme_gray()` 是默认主题。在像 `ggthemes` (<https://jrnold.github.io/ggthemes>) 这样的附加包中，Jeffrey Arnold 提供了更多的主题。如果你试图匹配特定的公司或期刊风格，你还可以创建自己的主题。

同样可以控制每个主题的各个组件，比如用于 y 轴的字体大小和颜色。我们已经知道 `legend.position` 控制图例绘制的位置。还有许多其他方面的图例可以用 `theme()` 函数进行定制。例如，在下面的图中，我们改变了图例的方向，并在其周围加上了黑色边框。请注意，主题的图例框和图形标题元素的定制是通过 `element_*` 函数完成的。这些函数指定非数据组件的样式，例如，在 `element_text()` 的 `face` 参数中，标题文本被加粗，图例边框颜色在 `element_rect()` 的 `color` 参数中定义。控制标题和标题位置的主题元素分别是 `plot.title.position` 和 `plot.caption.position`。在下面的图中，这些被设置为“plot”，以表示这些元素与整个绘图区域对齐，而不是与绘图面板（默认值）对齐。其他一些有用的 `theme()` 组件用于更改标题和标题文本格式的放置。

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  labs(
    title = "Larger engine sizes tend to have lower fuel economy",
    caption = "Source: 
  \) +
  theme\(
    legend.position = c\(0.6, 0.7\),
    legend.direction = "horizontal",
    legend.box.background = element\_rect\(color = "black"\),
    plot.title = element\_text\(face = "bold"\),
    plot.title.position = "plot",
```

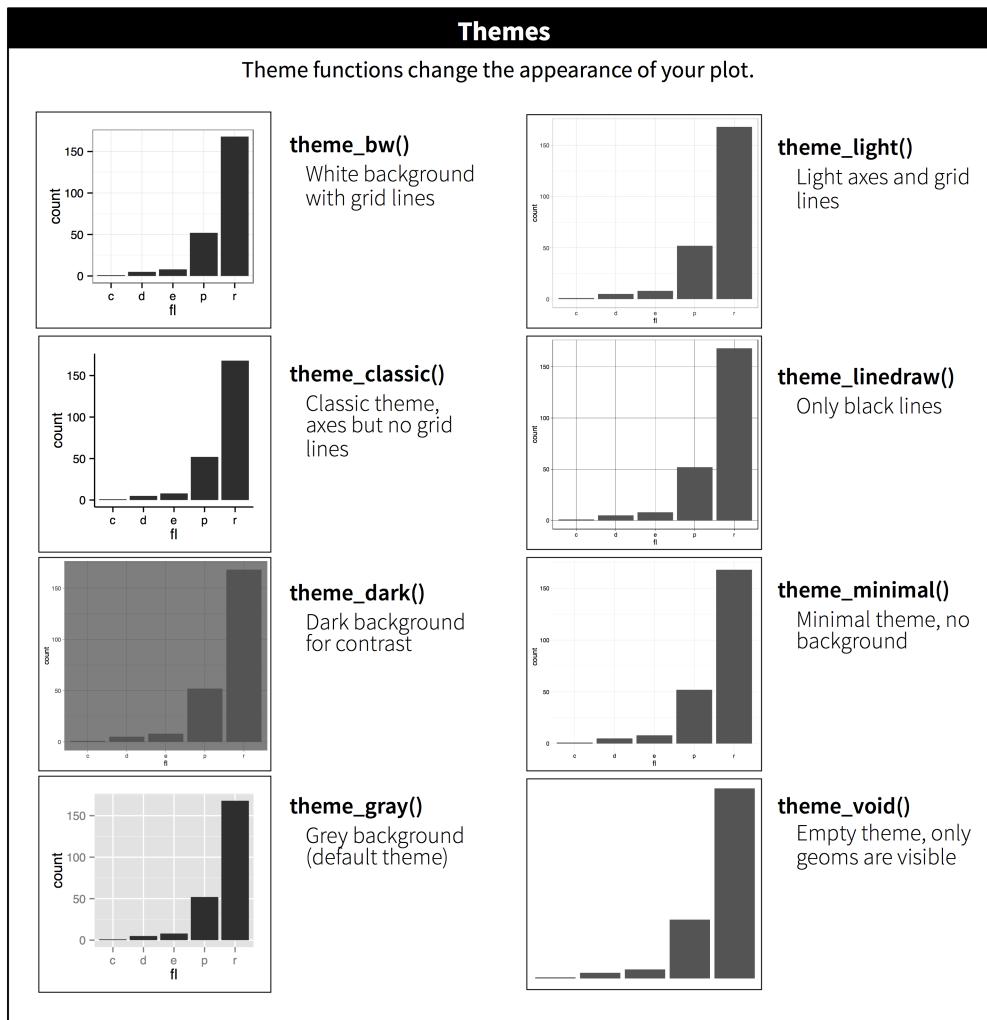
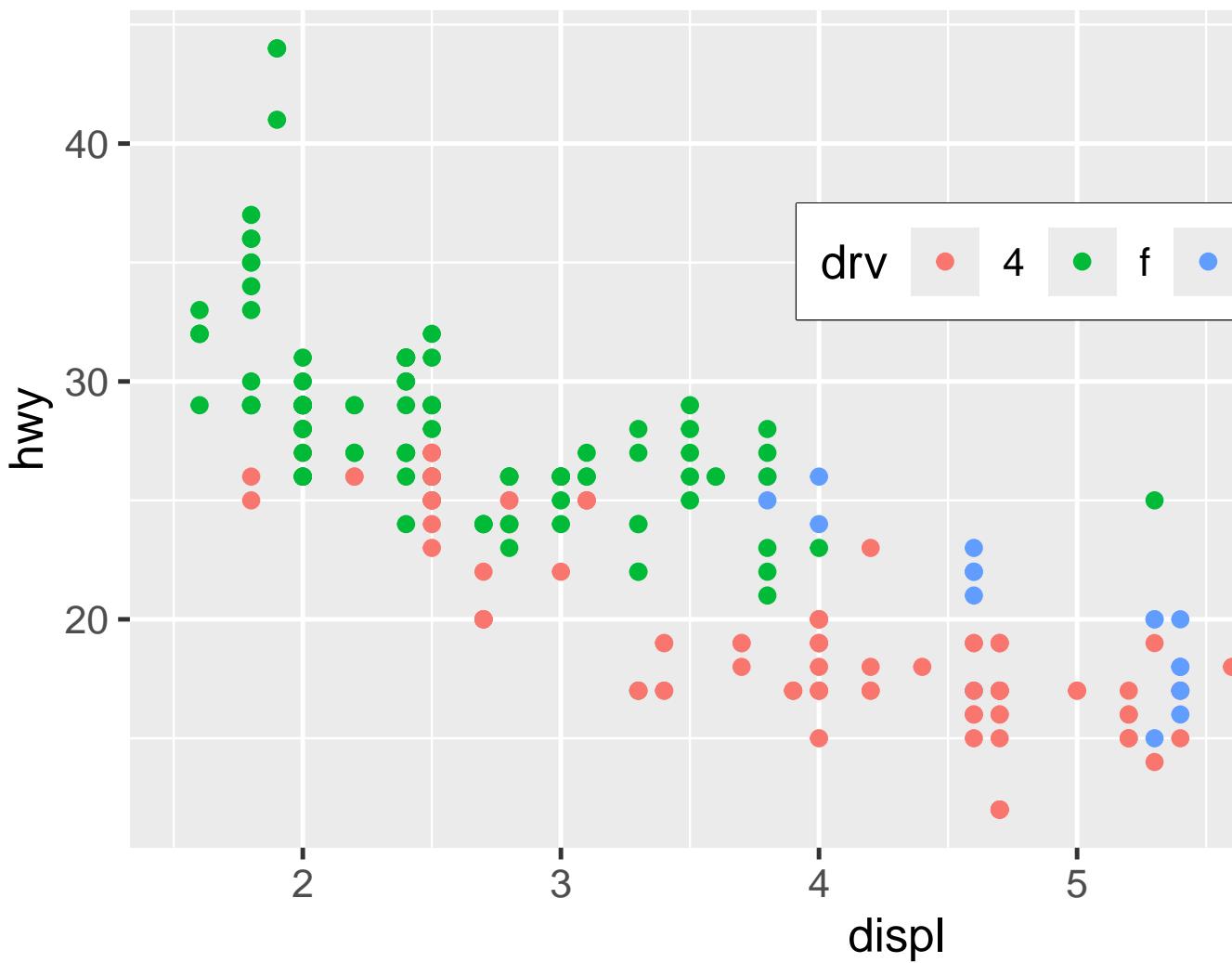


图 11.2: The eight themes built-in to ggplot2.

## 11 交流

```
plot.caption.position = "plot",
plot.caption = element_text(hjust = 0)
)
#> Warning: A numeric `legend.position` argument in `theme()` was deprecated in
#> 3.5.0.
#> i Please use the `legend.position.inside` argument of `theme()` instead.
```

## Larger engine sizes tend to have lower fuel economy



Source: <https://fueleconomy.gov>.

## 11 交流

要了解 `theme()` 函数的所有组件的概述，请查看`?theme` 的帮助文档。[ggplot2 book](#) 也是了解主题化完整细节的好地方。

For an overview of all `theme()` components, see help with `?theme`. The [ggplot2 book](#) is also a great place to go for the full details on theming.

### 11.5.1 练习

1. 从 `ggthemes` 包中选择一个主题，并将其应用到你最后制作的图形上。
2. 将你的图形的轴标签设置为蓝色并加粗。

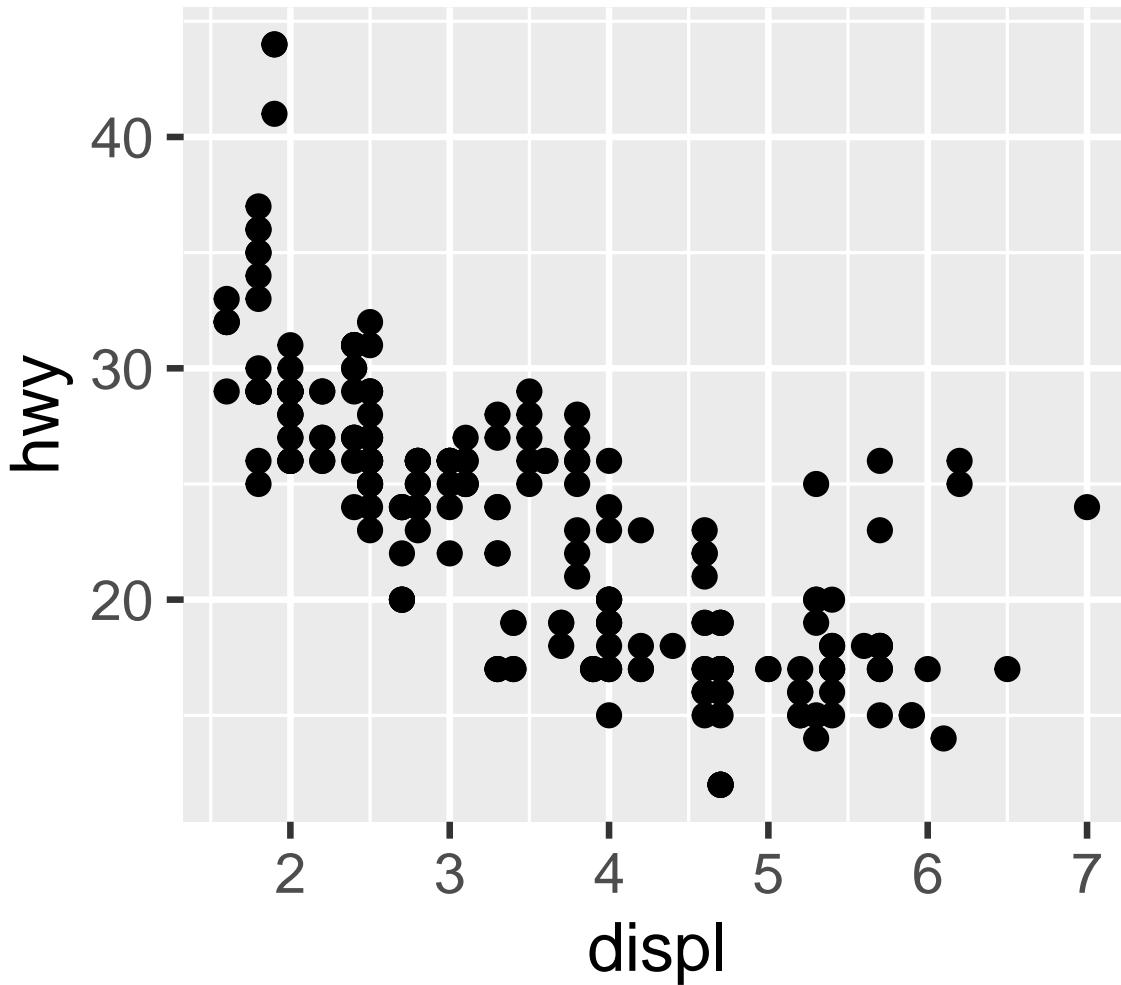
## 11.6 布局

到目前为止，我们讨论了如何创建和修改单个图形。但是，如果你有多个图形，并希望以某种方式将它们排列在一起怎么办？`patchwork` 包允许你将单独的图形组合到同一个图形中。在本章的前面部分，我们已经加载了这个包。

要将两个图形并排放置，你只需将它们加在一起。请注意，你首先需要创建图形并将它们保存为对象（在以下示例中，它们被称为 `p1` 和 `p2`）。然后，你使用 `+` 将它们并排放置。

```
p1 <- ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point() +  
  labs(title = "Plot 1")  
p2 <- ggplot(mpg, aes(x = drv, y = hwy)) +  
  geom_boxplot() +  
  labs(title = "Plot 2")  
p1 + p2
```

Plot 1



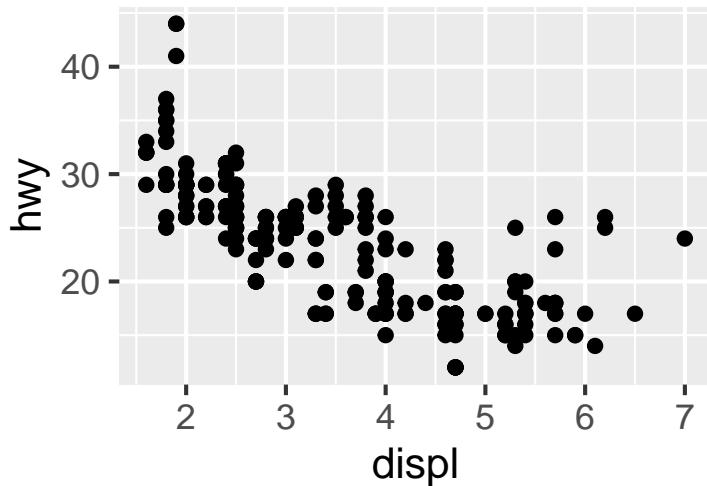
## 11 交流

需要注意的是，在上面的代码块中，我们并没有使用 `patchwork` 包中的新函数。相反，该包为 `+` 运算符添加了新的功能。

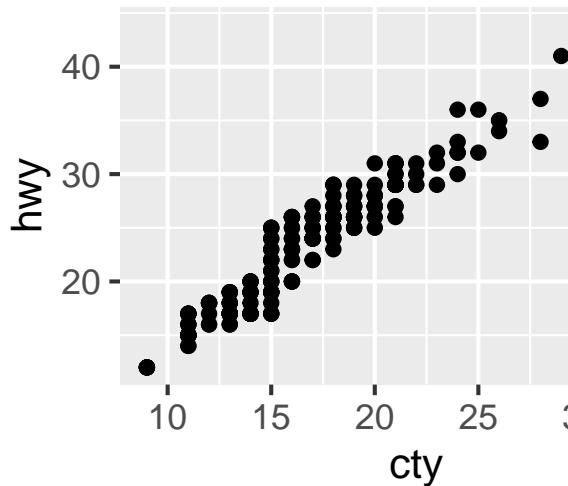
你还可以使用 `patchwork` 创建复杂的图形布局。在以下示例中，`|` 将 `p1` 和 `p3` 并排放置，而`/`将 `p2` 移动到下一行。

```
p3 <- ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point() +
  labs(title = "Plot 3")
(p1 | p3) / p2
```

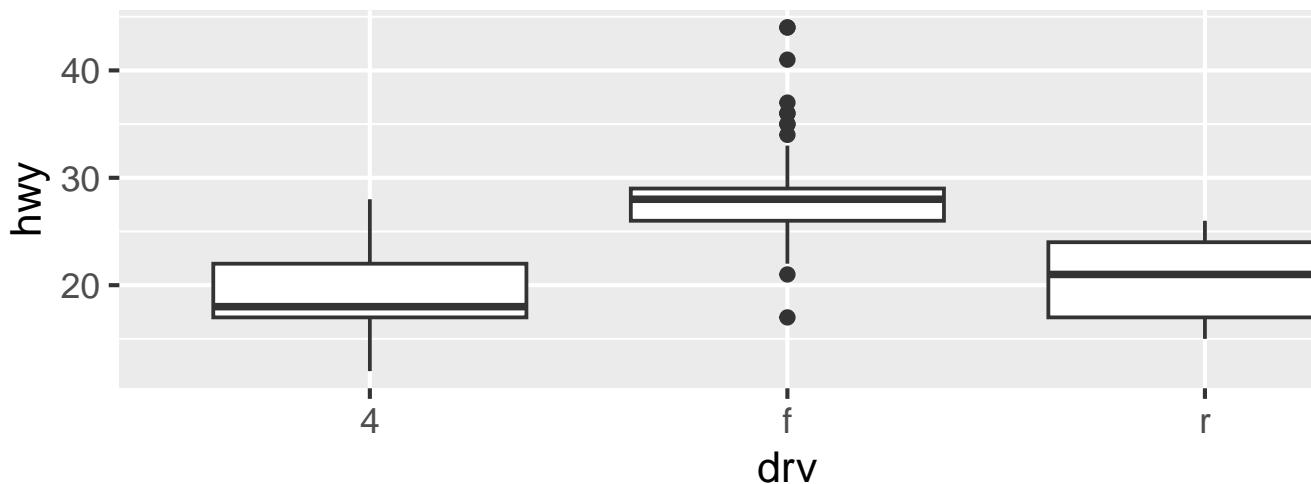
Plot 1



Plot 3



Plot 2



## 11 交流

此外，`patchwork` 允许你将多个图形的图例收集到一个共同的图例中，自定义图例的位置以及图形的尺寸，并为你的图形添加共同的标题、副标题、说明等。下面我们创建了 5 个图形。我们关闭了箱形图和散点图的图例，并将密度图的图例收集到图表顶部，使用了 `& theme(legend.position = "top")`。请注意这里使用了 `&` 运算符而不是通常的 `+`。这是因为我们正在修改 `patchwork` 图形的主题，而不是单个的 `ggplot`。图例被放置在顶部的 `guide_area()` 内。最后，我们还自定义了 `patchwork` 中各个组件的高度：指南的高度为 1，箱形图的高度为 3，密度图的高度为 2，分面的散点图的高度为 4。`patchwork` 使用这个比例来划分你为图形分配的区域，并相应地放置各个组件。

```
p1 <- ggplot(mpg, aes(x = drv, y = cty, color = drv)) +
  geom_boxplot(show.legend = FALSE) +
  labs(title = "Plot 1")

p2 <- ggplot(mpg, aes(x = drv, y = hwy, color = drv)) +
  geom_boxplot(show.legend = FALSE) +
  labs(title = "Plot 2")

p3 <- ggplot(mpg, aes(x = cty, color = drv, fill = drv)) +
  geom_density(alpha = 0.5) +
  labs(title = "Plot 3")

p4 <- ggplot(mpg, aes(x = hwy, color = drv, fill = drv)) +
  geom_density(alpha = 0.5) +
  labs(title = "Plot 4")

p5 <- ggplot(mpg, aes(x = cty, y = hwy, color = drv)) +
  geom_point(show.legend = FALSE) +
```

## 11.6 布局

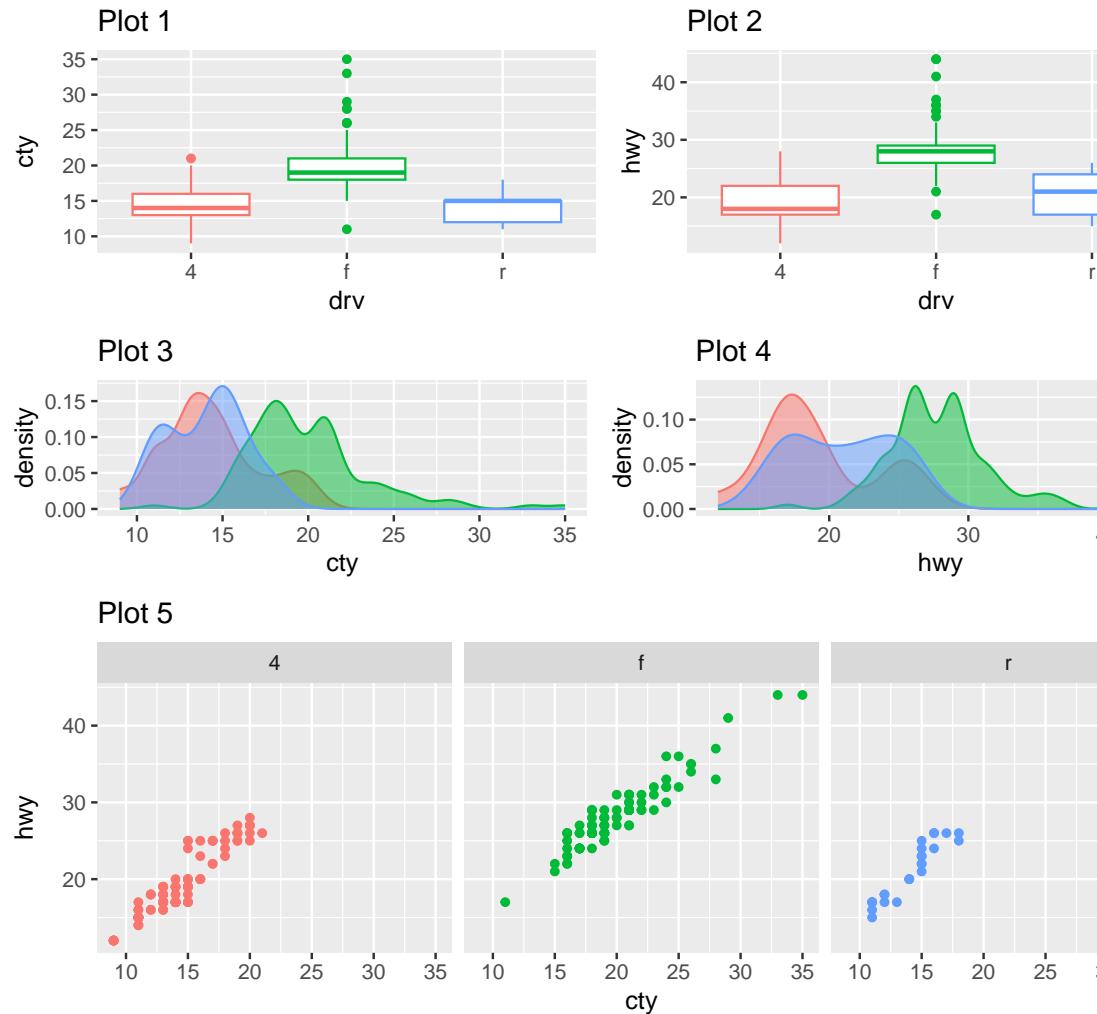
```
facet_wrap(~drv) +
  labs(title = "Plot 5")

(guide_area() / (p1 + p2) / (p3 + p4) / p5) +
  plot_annotation(
    title = "City and highway mileage for cars with different drive trains",
    caption = "Source: 
  \) +
  plot\_layout\(
    guides = "collect",
    heights = c\(1, 3, 2, 4\)
  \) &
  theme\(legend.position = "top"\)
```

## 11 交流

City and highway mileage for cars with different drive trains

drv 4 f r



Source: <https://fuelecon>

如果你希望了解更多关于如何使用 `patchwork` 组合和布局多个图形的信息，我们推荐你浏览该包的官方网站上的指南：<https://patchwork.data-imaginist.com>。

### 11.6.1 练习

1. 如果在下面的绘图布局中省略括号会发生什么？你能解释一下为什么会这样吗？

```
p1 <- ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point() +  
  labs(title = "Plot 1")  
p2 <- ggplot(mpg, aes(x = drv, y = hwy)) +  
  geom_boxplot() +  
  labs(title = "Plot 2")  
p3 <- ggplot(mpg, aes(x = cty, y = hwy)) +  
  geom_point() +  
  labs(title = "Plot 3")  
  
(p1 | p2) / p3
```

2. 使用前面练习中的三个图，重新创建下面的 `patchwork`。

Fig. A:

Plot 1

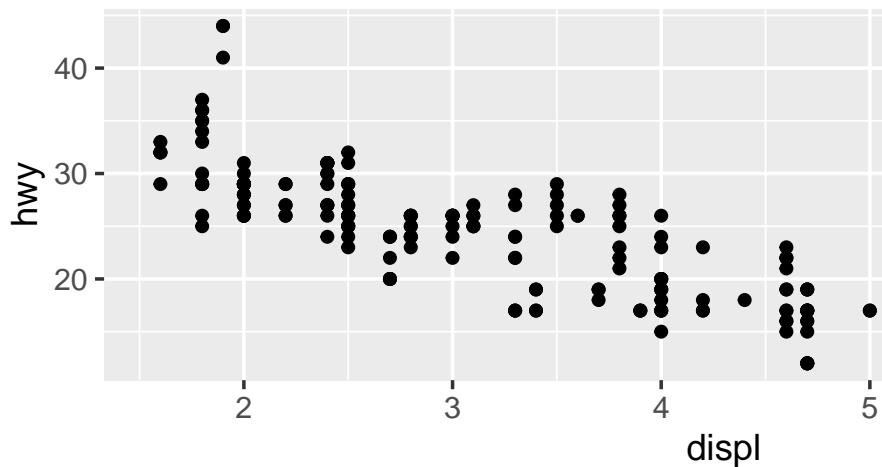


Fig. B:

Plot 2

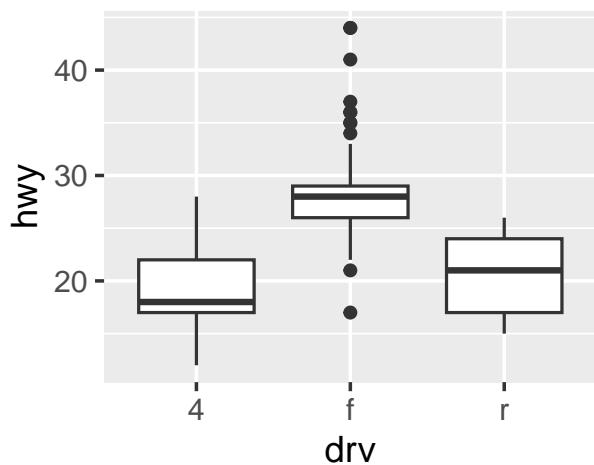
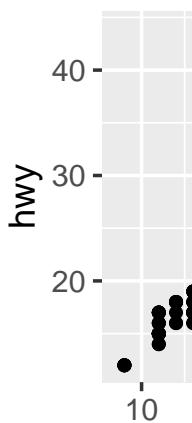


Fig. C:

Plot 3



## 11.7 小结

在本章中，你学习了如何添加图形标签，如标题、副标题、说明以及修改默认的轴标签，使用注释在图形中添加信息性文本或突出显示特定的数据点，自定义轴刻度，并更改图形的主题。你还学习了如何使用简单和复杂的图形布局将多个图形组合成一个图形。

虽然到目前为止，您已经学习了如何制作许多不同类型的图形以及如何使用各种技术来定制它们，但我们只是触及了使用 `ggplot2` 创建内容的冰山一角。如果你想全面了解 `ggplot2`，我们推荐您阅读《*ggplot2: Elegant Graphics for Data Analysis*》这本书。其他有用的资源还有 Winston Chang 的《*R Graphics Cookbook*》和 Claus Wilke 的《*Fundamentals of Data Visualization*》。



## **Part III**

### **转换**



本书的第二部分深入探讨了数据可视化。在这一部分，你将学习在数据框中最常遇到的变量类型，并学习可用于处理这些变量的工具。

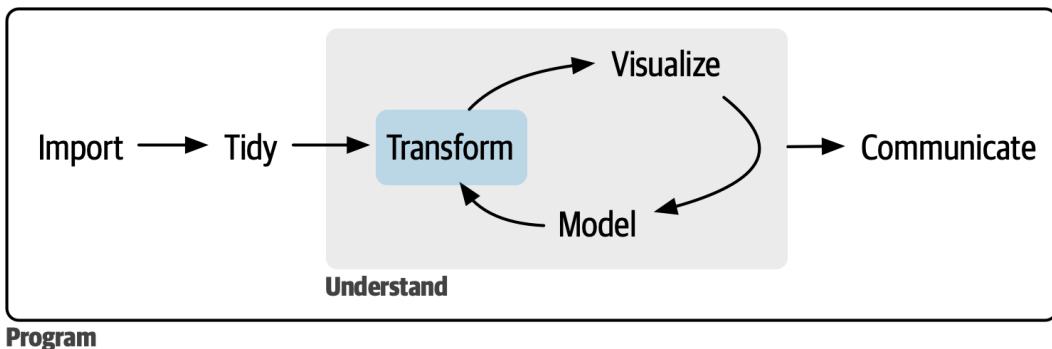


图 11.3: The options for data transformation depends heavily on the type of data involved, the subject of this part of the book.

你可以根据需要阅读这些章节。它们的设计在很大程度上是独立的，因此可以按顺序阅读，也可以不按顺序阅读。

- 章节 ?? 将向你介绍逻辑向量。逻辑向量是最简单的向量类型，但功能非常强大。你将学习如何通过数值比较创建它们，如何使用布尔代数将它们组合在一起，如何在汇总中使用它们，以及如何使用它们进行条件转换；
- 章节 ?? 将深入探讨数字向量的工具，这是数据科学的动力源泉。你将更深入地了解计数和一系列重要的转换和汇总函数；
- 章节 ?? 将为你提供处理字符串的工具。你可以切割它们，将它们分割成小块，然后再将它们重新组合在一起。这一章主要关注 stringr 包，但你也将学习一些更多用于从字符串中提取数据的 tidyverse 函数；
- 章节 ?? 将向你介绍正则表达式，这是一种强大的字符串操作工具。本章将带你从想象一只猫走过你的键盘，到阅读和编写复杂的字符串模式；
- 章节 ?? 将介绍因子，一种 R 用于存储分类数据的数据类型。当变量具有

一组固定的可能值时，或者当你想要使用非字母顺序的字符串排序时，你可以使用因子；

- 章节 ?? 将为你提供处理日期和日期时间的关键工具。不幸的是，你越了解日期时间，它们似乎就变得越复杂，但在 lubridate 包的帮助下，你将学习如何克服最常见的挑战；
- 章节 ?? 将深入探讨缺失值。我们之前已经单独讨论过它们几次了，但现在是时候全面讨论它们了，帮助你理解隐式和显式缺失值之间的区别，以及如何和为什么可在它们之间进行转换；
- 章节 ?? 通过为你提供将两个（或多个）数据框合并在一起的工具来结束本书的这一部分。学习合并（join）将迫使你了解键（keys）的概念，并思考如何标识数据集的每一行。

# 12 逻辑向量

## 12.1 引言

在本章中，你将学习处理逻辑向量的工具。逻辑向量是最简单的向量类型，因为每个元素只能是三个可能值之一：TRUE（真）、FALSE（假）和 NA（缺失值）。在原始数据中直接找到逻辑向量的情况相对较少，但在几乎每次分析的过程中，你都会创建和操作它们。

我们将首先讨论创建逻辑向量的最常见方法：通过数值比较。然后，你将学习如何使用布尔代数来组合不同的逻辑向量，以及一些有用的总结。最后，我们将介绍 `if_else()` 和 `case_when()` 这两个由逻辑向量驱动的用于进行条件更改的有用函数。

### 12.1.1 必要条件

本章中你将学习的大部分函数都是由基础 R 提供的，因此我们不需要 tidyverse，但我们仍然会加载它，以便我们可以使用 `mutate()`、`filter()` 等函数来处理数据框。我们还将继续从 `nycflights13::flights` 数据集中提取示例。

## 12 逻辑向量

```
library(tidyverse)
library(nycflights13)
```

然而，随着我们开始介绍更多的工具，并不总是会有一个完美的真实示例。因此，我们将开始使用 `c()` 创建一些虚拟数据：

```
x <- c(1, 2, 3, 5, 7, 11, 13)
x * 2
#> [1] 2 4 6 10 14 22 26
```

这样做使得解释单个函数变得更容易，但代价是使得它较难应用于你的数据问题。请记住，我们对自由浮动的向量所做的任何操作，你都可以通过 `mutate()` 等函数对数据框中的变量执行相同的操作。

```
df <- tibble(x)
df |>
  mutate(y = x * 2)
#> # A tibble: 7 x 2
#>       x     y
#>   <dbl> <dbl>
#> 1     1     2
#> 2     2     4
#> 3     3     6
#> 4     5    10
#> 5     7    14
#> 6    11    22
#> # i 1 more row
```

## 12.2 比较

创建逻辑向量的一个非常常见的方式是通过使用 `<`、`<=`、`>`、`>=`、`!=` 和 `==` 进行数值比较。到目前为止，我们主要在 `filter()` 内部临时创建逻辑变量，它们被计算、使用，然后就被丢弃了。例如，下面的 `filter()` 查找所有大致准时到达的日间航班：

```
flights |>
  filter(dep_time > 600 & dep_time < 2000 & abs(arr_delay) < 20)
#> # A tibble: 172,286 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       601         600        1     844        850
#> 2 2013     1     1       602         610       -8     812        820
#> 3 2013     1     1       602         605       -3     821        805
#> 4 2013     1     1       606         610       -4     858        910
#> 5 2013     1     1       606         610       -4     837        845
#> 6 2013     1     1       607         607        0     858        915
#> # i 172,280 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

知道这是一种快捷方式，并且可以使用 `mutate()` 明确创建底层的逻辑变量是很有用的：

```
flights |>
  mutate(
    daytime = dep_time > 600 & dep_time < 2000,
    approx_ontime = abs(arr_delay) < 20,
```

## 12 逻辑向量

```
.keep = "used"
)
#> # A tibble: 336,776 x 4
#>   dep_time arr_delay daytime approx_ontime
#>   <int>     <dbl> <lgl>    <lgl>
#> 1     517      11 FALSE    TRUE
#> 2     533      20 FALSE    FALSE
#> 3     542      33 FALSE    FALSE
#> 4     544     -18 FALSE    TRUE
#> 5     554     -25 FALSE    FALSE
#> 6     554      12 FALSE    TRUE
#> # i 336,770 more rows
```

这在处理更复杂的逻辑时特别有用，因为命名中间步骤既可以让代码更容易读，也更容易检查每个步骤是否正确计算。

总的来说，最初的 `filter` 函数相当于：

```
flights |>
  mutate(
    daytime = dep_time > 600 & dep_time < 2000,
    approx_ontime = abs(arr_delay) < 20,
  ) |>
  filter(daytime & approx_ontime)
```

### 12.2.1 浮点数比较

对数字慎用 `==`。例如，下面这个向量看起来包含数字 1 和 2：

## 12.2 比较

```
x <- c(1 / 49 * 49, sqrt(2) ^ 2)
x
#> [1] 1 2
```

但如果你测试它们是否相等，你得到 FALSE:

```
x == c(1, 2)
#> [1] FALSE FALSE
```

这是怎么回事呢？计算机存储数字时只有固定的小数位数，因此无法精确表示  $1/49$  或 `sqrt(2)`，随后的计算将略有偏差。我们可以通过在 `print()` 函数中指定 `digits`<sup>1</sup> 参数来查看精确值：

```
print(x, digits = 16)
#> [1] 0.9999999999999999 2.0000000000000004
```

你可以看到为什么 R 默认会对这些数字进行四舍五入；它们确实非常接近你预期的值。

既然你已经明白了为什么 `==` 会失败，那你能做什么呢？一个选择是使用 `dplyr::near()` 函数，它会忽略微小的差异：

```
near(x, c(1, 2))
#> [1] TRUE TRUE
```

---

<sup>1</sup>R 通常会自动为你调用 `print` 函数（即 `x` 是 `print(x)` 的简写），但如果你想要提供其他参数，显式调用 `print` 函数会很有用。

### 12.2.2 缺失值

缺失值代表未知，因此它们是“具有传染性的”：几乎任何涉及未知值的操作也将是未知的：

```
NA > 5
#> [1] NA
10 == NA
#> [1] NA
```

最令人困惑的结果是这一个：

```
NA == NA
#> [1] NA
```

如果我们人为地添加一些额外的上下文，将更容易理解为什么这是正确的：

```
# We don't know how old Mary is
age_mary <- NA

# We don't know how old John is
age_john <- NA

# Are Mary and John the same age?
age_mary == age_john
#> [1] NA
# We don't know!
```

所以，如果你想要找出所有 `dep_time` 缺失的航班，以下代码是不起作用的，因为 `dep_time == NA` 对于每一行都会返回 `NA`，而 `filter()` 函数会自动丢弃缺失值：

```
flights |>
  filter(dep_time == NA)
#> # A tibble: 0 x 19
#> # i 19 variables: year <int>, month <int>, day <int>, dep_time <int>,
#> #   sched_dep_time <int>, dep_delay <dbl>, arr_time <int>, ...
```

相反，我们需要一个新工具：`is.na()`。

### 12.2.3 `is.na()`

`is.na(x)` 适用于任何类型的向量，对于缺失值返回 `TRUE`，对于其他所有值返回 `FALSE`。

```
is.na(c(TRUE, NA, FALSE))
#> [1] FALSE TRUE FALSE
is.na(c(1, NA, 3))
#> [1] FALSE TRUE FALSE
is.na(c("a", NA, "b"))
#> [1] FALSE TRUE FALSE
```

我们可以使用 `is.na()` 来找到所有缺失 `dep_time` 的行：

## 12 逻辑向量

```
flights |>
  filter(is.na(dep_time))
#> # A tibble: 8,255 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       NA           1630        NA        NA          1815
#> 2 2013     1     1       NA           1935        NA        NA          2240
#> 3 2013     1     1       NA           1500        NA        NA          1820
#> 4 2013     1     1       NA            600        NA        NA          900
#> 5 2013     1     2       NA           1540        NA        NA          1740
#> 6 2013     1     2       NA           1620        NA        NA          1740
#> # i 8,249 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

`is.na()` 在 `arrange()` 函数中也非常有用。`arrange()` 通常将所有缺失值放在最后，但你可以通过首先根据 `is.na()` 进行排序来覆盖这个默认行为：

```
flights |>
  filter(month == 1, day == 1) |>
  arrange(dep_time)
#> # A tibble: 842 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          810
#> 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         1020
```

## 12.2 比较

```
#> 5 2013     1     1      554          600       -6      812      837
#> 6 2013     1     1      554          558       -4      740      728
#> # i 836 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
flights |>
  filter(month == 1, day == 1) |>
  arrange(desc(is.na(dep_time)), dep_time)
#> # A tibble: 842 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       NA         1630       NA       NA        1815
#> 2 2013     1     1       NA         1935       NA       NA        2240
#> 3 2013     1     1       NA         1500       NA       NA        1825
#> 4 2013     1     1       NA          600       NA       NA        901
#> 5 2013     1     1       517         515        2      830        819
#> 6 2013     1     1       533         529        4      850        830
#> # i 836 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

我们将在章节 ?? 更深入地讨论缺失值。

### 12.2.4 练习

1. `dplyr::near()` 是如何工作的? 键入 `near` 以查看源代码。`sqrt(2)^2` 与 2 接近吗?

2. 结合使用 `mutate()`, `is.na()`, 和 `count()` 来描述 `dep_time`、`sched_dep_time` 和 `dep_delay` 中缺失值之间的联系。

## 12.3 布尔代数

若你有了多个逻辑向量，可以使用布尔代数将它们组合在一起。在 R 中，`&` 是“和”，`|` 是“或”，`!` 是“非”，而 `xor()` 是异或<sup>2</sup>。例如，`df %>% filter(!is.na(x))` 查找所有 `x` 不缺失的行，而 `df %>% filter(x < -10 | x > 0)` 查找所有 `x` 小于-10 或大于 0 的行。图 ?? 展示了完整的布尔操作集以及它们是如何工作的。

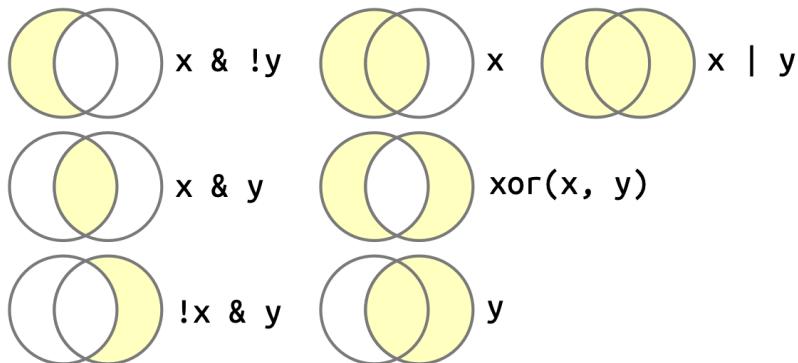


图 12.1: The complete set of Boolean operations. `x` is the left-hand circle, `y` is the right-hand circle, and the shaded region show which parts each operator selects.

除了 `&` 和 `|` 之外，R 还有 `&&` 和 `||`。不要在 `dplyr` 函数中使用它们！这些被称为短路运算符，并且总是只返回一个 `TRUE` 或 `FALSE`。它们对编程很重

<sup>2</sup>也就是说，如果 `x` 为真，或 `y` 为真，但两者不同时为真，那么 `xor(x, y)` 就为真。这通常是我们用英语中使用“或”的方式。“两者都”通常不是对“你想要冰淇淋还是蛋糕？”这个问题的可接受答案。

要，但对数据科学来说并不重要。

### 12.3.1 缺失值

布尔代数中缺失值的规则有点难解释，因为乍一看它们似乎不一致：

```
df <- tibble(x = c(TRUE, FALSE, NA))

df |>
  mutate(
    and = x & NA,
    or = x | NA
  )

#> # A tibble: 3 x 3
#>   x     and    or
#>   <lgl> <lgl> <lgl>
#> 1 TRUE  NA    TRUE
#> 2 FALSE FALSE NA
#> 3 NA    NA    NA
```

要理解其中的原因，可以考虑 `NA | TRUE`（即 `NA` 或 `TRUE`）。在逻辑向量中，缺失值意味着该值可能是 `TRUE` 或 `FALSE`。因为至少有一个是 `TRUE`，所以 `TRUE | TRUE` 和 `FALSE | TRUE` 都是 `TRUE`。`NA | TRUE` 也必须是 `TRUE`，因为 `NA` 可以是 `TRUE` 或 `FALSE`。然而，`NA | FALSE` 是 `NA`，因为我们不知道 `NA` 是 `TRUE` 还是 `FALSE`。类似的推理也适用于 `NA & FALSE`。

### 12.3.2 运算顺序

请注意，操作的顺序并不与英语中的顺序相同。以下代码用于查找所有在十一月或十二月出发的航班：

```
flights |>
  filter(month == 11 | month == 12)
```

你可能会想要像在英语中那样写：“Find all flights that departed in November or December.”：

```
flights |>
  filter(month == 11 | 12)
#> # 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     1      517           515       2     830           813
#> 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           830
#> 6 2013     1     1      554           558      -4     740           728
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

这段代码没有报错，但看起来也没有正常工作。这是怎么回事呢？在这里，R 首先计算 `month == 11` 创建一个逻辑向量，我们称之为 `nov`。然后它计算 `nov | 12`。当你使用数字与逻辑运算符结合时，除了 0 以外的所有数字都会被

转换为 TRUE，因此这等价于 nov | TRUE，而 TRUE | TRUE 总是 TRUE，所以每一行都会被选中：

```
flights |>
  mutate(
    nov = month == 11,
    final = nov | 12,
    .keep = "used"
  )
#> # A tibble: 336,776 x 3
#>   month nov   final
#>   <int> <lgl> <lgl>
#> 1     1 FALSE TRUE
#> 2     1 FALSE TRUE
#> 3     1 FALSE TRUE
#> 4     1 FALSE TRUE
#> 5     1 FALSE TRUE
#> 6     1 FALSE TRUE
#> # i 336,770 more rows
```

### 12.3.3 %in%

避免 ==s 和 !s 顺序错误的一个简单方法是使用%in%。x %in% y 返回一个与 x 长度相同的逻辑向量，当 x 中的某个值在 y 中时，该逻辑向量的对应位置为 TRUE。

## 12 逻辑向量

```
1:12 %in% c(1, 5, 11)
#> [1] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE
letters[1:10] %in% c("a", "e", "i", "o", "u")
#> [1] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE
```

因此，要查找 11 月和 12 月的所有航班，可以这样写：

```
flights |>
  filter(month %in% c(11, 12))
```

请注意，`%in%` 对 NA 的处理规则与 `==` 不同，因为 `NA %in% NA` 是 `TRUE`。

Note that `%in%` obeys different rules for NA to `==`, as `NA %in% NA` is `TRUE`.

```
c(1, 2, NA) == NA
#> [1] NA NA NA
c(1, 2, NA) %in% NA
#> [1] FALSE FALSE TRUE
```

这可以作为一个有用的快捷方式：

```
flights |>
  filter(dep_time %in% c(NA, 0800))
#> # A tibble: 8,803 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>
```

## 12.4 小结

```
#> 1 2013 1 1 800 800 0 1022 1014
#> 2 2013 1 1 800 810 -10 949 955
#> 3 2013 1 1 NA 1630 NA NA 1815
#> 4 2013 1 1 NA 1935 NA NA 2240
#> 5 2013 1 1 NA 1500 NA NA 1825
#> 6 2013 1 1 NA 600 NA NA 901
#> # i 8,797 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

### 12.3.4 练习

- 找出所有到达延误 (`arr_delay`) 缺失但起飞延误 (`dep_delay`) 不缺失的航班。找出所有到达时间 (`arr_time`) 和计划到达时间 (`sched_arr_time`) 都不缺失，但到达延误 (`arr_delay`) 缺失的航班。
- 有多少航班的起飞时间 (`dep_time`) 是缺失的？这些行中还缺失了哪些其他变量？这些行可能代表什么？
- 假设缺失的起飞时间 (`dep_time`) 意味着航班被取消，查看每天取消的航班数量。是否存在某种模式？被取消航班的比例与非】未被取消航班的平均延误之间是否存在联系？

## 12.4 小结

以下部分描述了一些用于总结逻辑向量的有用技术。除了专门与逻辑向量一起工作的函数外，你还可以使用与数字向量一起工作的函数。

### 12.4.1 逻辑汇总函数 (logical summaries)

有两个主要的逻辑汇总函数: `any()` 和 `all()`。`any(x)` 相当于 `|`, 如果 `x` 中有任何 `TRUE` 值, 它就会返回 `TRUE`。`all(x)` 相当于 `&`, 只有当 `x` 的所有值都是 `TRUE` 时, 它才会返回 `TRUE`。像所有汇总函数一样, 如果存在任何缺失值, 它们将返回 `NA`, 并且你可以像往常一样使用 `na.rm = TRUE` 来让缺失值消失。

例如, 我们可以使用 `all()` 和 `any()` 来找出是否每个航班的起飞延误都不超过一个小时, 或者是否有任何航班的到达延误了五个小时或更多。使用 `group_by()` 可以让我们按天来执行这些操作:

```
flights |>
  group_by(year, month, day) |>
  summarize(
    all_delayed = all(dep_delay <= 60, na.rm = TRUE),
    any_long_delay = any(arr_delay >= 300, na.rm = TRUE),
    .groups = "drop"
  )
#> # A tibble: 365 x 5
#>   year month   day all_delayed any_long_delay
#>   <int> <int> <int> <lgl>        <lgl>
#> 1  2013     1     1 FALSE      TRUE
#> 2  2013     1     2 FALSE      TRUE
#> 3  2013     1     3 FALSE     FALSE
#> 4  2013     1     4 FALSE     FALSE
#> 5  2013     1     5 FALSE      TRUE
#> 6  2013     1     6 FALSE     FALSE
#> # i 359 more rows
```

然而，在大多数情况下，`any()` 和 `all()` 有点过于粗略，如果能够更详细地了解有多少值是 `TRUE` 或 `FALSE` 就好了。这就引出了数值摘要（numeric summaries）。

### 12.4.2 逻辑向量的数值汇总

当你在数值上下文中使用逻辑向量时，`TRUE` 变成 1，`FALSE` 变成 0。这使得 `sum()` 和 `mean()` 函数对逻辑向量非常有用，因为 `sum(x)` 给出 `TRUE` 的数量，而 `mean(x)` 给出 `TRUE` 的比例（因为 `mean()` 实际上就是 `sum()` 除以 `length()`）。

例如，我们可以查看起飞延误最多一小时的航班的比例，以及到达延误五小时或更多的航班的数量：

```
flights |>
  group_by(year, month, day) |>
  summarize(
    proportion_delayed = mean(dep_delay <= 60, na.rm = TRUE),
    count_long_delay = sum(arr_delay >= 300, na.rm = TRUE),
    .groups = "drop"
  )
#> # A tibble: 365 x 5
#>   year month   day proportion_delayed count_long_delay
#>   <int> <int> <int>             <dbl>           <int>
#> 1  2013     1     1             0.939            3
#> 2  2013     1     2             0.914            3
#> 3  2013     1     3             0.941            0
#> 4  2013     1     4             0.953            0
```

## 12 逻辑向量

```
#> 5 2013 1 5 0.964 1  
#> 6 2013 1 6 0.959 0  
#> # i 359 more rows
```

### 12.4.3 逻辑子集 (logical subsetting)

逻辑向量在汇总中的最后一个用途是：你可以使用逻辑向量来筛选单个变量到感兴趣的子集。这利用了基本的子集操作符 `[` (发音为 “subset”)，你将在小节 `??` 中学到更多关于它的内容。

我们想要查看实际上有延误的航班的平均延误时间。一种方法是先筛选航班，然后计算平均延误时间：

```
flights |>  
  filter(arr_delay > 0) |>  
  group_by(year, month, day) |>  
  summarize(  
    behind = mean(arr_delay),  
    n = n(),  
    .groups = "drop"  
  )  
#> # A tibble: 365 x 5  
#>   year month day behind     n  
#>   <int> <int> <int>  <dbl> <int>  
#> 1 2013    1     1    32.5    461  
#> 2 2013    1     2    32.0    535  
#> 3 2013    1     3    27.7    460  
#> 4 2013    1     4    28.3    297
```

## 12.4 小结

```
#> 5 2013     1     5  22.6  238
#> 6 2013     1     6  24.4  381
#> # i 359 more rows
```

这样做是可以的，但如果我们还想计算提前到达的航班的平均延误时间呢？我们需要执行一个单独的筛选步骤，然后考虑如何将两个数据框合并在一起。相反，你可以使用 [来执行内联筛选：`arr_delay[arr_delay > 0]` 将仅返回正的到达延误时间。

由此引出：

```
flights |>
  group_by(year, month, day) |>
  summarize(
    behind = mean(arr_delay[arr_delay > 0], na.rm = TRUE),
    ahead = mean(arr_delay[arr_delay < 0], na.rm = TRUE),
    n = n(),
    .groups = "drop"
  )
#> # A tibble: 365 x 6
#>   year month   day behind ahead     n
#>   <int> <int> <int>  <dbl> <dbl> <int>
#> 1 2013     1     1   32.5 -12.5   842
#> 2 2013     1     2   32.0 -14.3   943
#> 3 2013     1     3   27.7 -18.2   914
#> 4 2013     1     4   28.3 -17.0   915
#> 5 2013     1     5   22.6 -14.0   720
```

## 12 逻辑向量

```
#> 6 2013     1     6   24.4 -13.6   832
#> # i 359 more rows
```

同时也要注意组大小的差异：在第一部分中，`n()` 给出的是每天延误航班的数量；在第二部分中，`n()` 给出的是总航班数量。

### 12.4.4 练习

1. `sum(is.na(x))` 会告诉你什么信息？`mean(is.na(x))` 呢？
2. 当应用于逻辑向量时，`prod()` 返回什么？它等于什么逻辑汇总函数？当应用于逻辑向量时 `min()` 返回什么？它等于什么逻辑汇总函数？阅读文档并进行一些实验。

## 12.5 条件转换

逻辑向量最强大的功能之一是用于条件转换，即当满足条件 `x` 时执行一个操作，当满足条件 `y` 时执行另一个操作。有两个重要的工具可以实现这一功能：`if_else()` 和 `case_when()`。

### 12.5.1 if\_else()

如果你想要在一个条件为 `TRUE` 时使用一个值，而在条件为 `FALSE` 时使用另一个值，你可以使用 `dplyr::if_else()`。你总是会用到 `if_else()` 的前三个参数。第一个参数 `condition`，是一个逻辑向量；第二个参数 `true`，给出当条件为真时的输出；第三个参数 `false`，给出当条件为假时的输出。

## 12.5 条件转换

让我们从一个简单的例子开始, 将一个数值向量标记为 “+ve”(正)或 “-ve”(负):

```
x <- c(-3:3, NA)
if_else(x > 0, "+ve", "-ve")
#> [1] "-ve"  "-ve"  "-ve"  "-ve"  "+ve"  "+ve"  "+ve"  NA
```

还有一个可选的第四个参数, 如果输入是 `NA`, 则会使用 `missing`:

```
if_else(x > 0, "+ve", "-ve", "???")
#> [1] "-ve"  "-ve"  "-ve"  "-ve"  "+ve"  "+ve"  "+ve"  "???"
```

你还可以使用向量作为 `true` 和 `false` 参数。例如, 这允许我们创建一个 `abs()` 函数的简化实现:

```
if_else(x < 0, -x, x)
#> [1]  3  2  1  0  1  2  3 NA
```

到目前为止, 所有的参数都使用了相同的向量, 当然你也可以混合和匹配。例如, 你可以像这样实现 `coalesce()` 函数的简单版本:

```
x1 <- c(NA, 1, 2, NA)
y1 <- c(3, NA, 4, 6)
if_else(is.na(x1), y1, x1)
#> [1] 3 1 2 6
```

你可能已经注意到我们上面标记示例中的一个小瑕疵: 零既不是正数也不是负数。我们可以通过添加另一个 `if_else()` 来解决这个问题:

## 12 逻辑向量

```
if_else(x == 0, "0", if_else(x < 0, "-ve", "+ve"), "??")
#> [1] "-ve"  "-ve"  "-ve"  "0"   "+ve"  "+ve"  "+ve"  "???"
```

这已经有点难以阅读了，你可以想象如果有更多的条件，它只会变得更难。相反，你可以切换到 `dplyr::case_when()`。

### 12.5.2 `case_when()`

`dplyr` 的 `case_when()` 是受 SQL 的 CASE 语句启发，为不同条件执行不同计算提供了灵活的方法。它有一个特殊的语法，不过这个语法看起来与 `tidyverse` 中将要使用的任何其他内容都不相似。它接受形如 `condition ~ output` 的组合，`condition` 必须是一个逻辑向量，当它为 TRUE 时，将使用 `output`。

这意味着我们可以将之前嵌套的 `if_else()` 重新创建如下：

```
x <- c(-3:3, NA)
case_when(
  x == 0 ~ "0",
  x < 0 ~ "-ve",
  x > 0 ~ "+ve",
  is.na(x) ~ "??"
)
#> [1] "-ve"  "-ve"  "-ve"  "0"   "+ve"  "+ve"  "+ve"  "???"
```

虽然代码更多，但它也更清晰。

为了解释 `case_when()` 是如何工作的，让我们探索一些更简单的案例。如果没有任何情况匹配，则输出为 NA：

## 12.5 条件转换

```
case_when(  
  x < 0 ~ "-ve",  
  x > 0 ~ "+ve"  
)  
#> [1] "-ve"  "-ve"  "-ve"  NA      "+ve"  "+ve"  "+ve"  NA
```

如果你想为未匹配到任何条件的情况设置一个“默认”的值，可以使用`.default`:

```
case_when(  
  x < 0 ~ "-ve",  
  x > 0 ~ "+ve",  
  .default = "???"  
)  
#> [1] "-ve"  "-ve"  "-ve"  "???"  "+ve"  "+ve"  "+ve"  "???"
```

注意，如果多个条件匹配，只会使用第一个条件:

```
case_when(  
  x > 0 ~ "+ve",  
  x > 2 ~ "big"  
)  
#> [1] NA     NA     NA     NA      "+ve"  "+ve"  "+ve"  NA
```

就像使用`if_else()`一样，你可以在`~`的两边使用变量，并且可以根据需要混合和匹配变量来解决你的问题。例如，我们可以使用`case_when()`为到达延误提供一些人们可读的标签:

## 12 逻辑向量

```
flights |>
  mutate(
    status = case_when(
      is.na(arr_delay) ~ "cancelled",
      arr_delay < -30 ~ "very early",
      arr_delay < -15 ~ "early",
      abs(arr_delay) <= 15 ~ "on time",
      arr_delay < 60 ~ "late",
      arr_delay < Inf ~ "very late",
    ),
    .keep = "used"
  )
#> # A tibble: 336,776 x 2
#>   arr_delay status
#>       <dbl> <chr>
#> 1        11 on time
#> 2        20 late
#> 3        33 late
#> 4       -18 early
#> 5       -25 early
#> 6        12 on time
#> # i 336,770 more rows
```

在编写这种复杂的 `case_when()` 语句时要小心；我最初的两次尝试混合使用了 `<` 和 `>`，结果不小心创建了重叠的条件。

### 12.5.3 兼容类型 (compatible types)

请注意，`if_else()` 和 `case_when()` 都要求输出具有兼容的类型。如果它们不兼容，你会看到类似这样的错误：

```
if_else(TRUE, "a", 1)
#> Error in `if_else()`:
#> ! Can't combine `true` <character> and `false` <double>.

case_when(
  x < -1 ~ TRUE,
  x > 0 ~ now()
)
#> Error in `case_when()`:
#> ! Can't combine `..1 (right)` <logical> and `..2 (right)` <datetime<local>>.
```

总体来说，兼容的类型是相对较少的，因为自动将一个类型的向量转换为另一个类型的向量是常见的错误来源。以下是一些最重要的兼容情况：

- 数字和逻辑向量是兼容的，正如我们在 `@sec-numeric-summaries-of-logicals` 讨论的那样。
- 字符串和因子（章节 `??`）是兼容的，因为可以将因子视为一组具有固定值的字符串。
- 日期和日期-时间，我们将在章节 `??` 部分讨论，它们是兼容的，因为你可以在将日期视为日期-时间的一个特例。
- `NA`，从技术上讲是逻辑向量，与所有类型都兼容，因为每个向量都有表示缺失值的方式。

## 12 逻辑向量

我们不要求你记住这些规则，但随着时间的推移，它们应该变得自然而然，因为它们在整个 tidyverse 中都被一致地应用。

### 12.5.4 练习

1. 如果一个数能被 2 整除，那么它就是偶数。在 R 中，你可以通过 `x %% 2 == 0` 来判断一个数是否为偶数。利用这个事实和 `if_else()` 函数来确定 0 到 20 之间的每个数是偶数还是奇数。
2. 给定一个天数向量，如 `x <- c("Monday", "Saturday", "Wednesday")`，使用 `if_else()` 语句将它们标记为周末或工作日。
3. 使用 `if_else()` 函数计算一个名为 `x` 的数值向量的绝对值。
4. 编写一个 `case_when()` 语句，利用 `flights` 数据集中的 `month` 和 `day` 列来标记一些重要的美国节假日（例如，新年、7 月 4 日、感恩节和圣诞节）。首先创建一个逻辑列，其值为 `TRUE` 或 `FALSE`，然后创建一个字符列，该列要么给出节假日的名称，要么是 `NA`。

## 12.6 小结

逻辑向量的定义很简单，因为每个值都必须是 `TRUE`、`FALSE` 或 `NA`。但逻辑向量提供了巨大的功能。在本章中，你学习了如何使用 `>`、`<`、`<=`、`>=`、`==`、`!=` 和 `is.na()` 创建逻辑向量，如何使用 `!`、`&` 和 `|` 组合它们，以及如何使用 `any()`、`all()`、`sum()` 和 `mean()` 对它们进行汇总。你还学习了强大的 `if_else()` 和 `case_when()` 函数，这些函数允许你根据逻辑向量的值返回值。

## 12.6 小结

在接下来的章节中，我们将一次又一次地看到逻辑向量。例如，在 @sec-strings，你将学习 `str_detect(x, pattern)`，它返回一个逻辑向量，对于 x 中匹配模式的元素，该向量的值为 TRUE；在 @sec-dates-and-times，你将通过比较日期和时间来创建逻辑向量。但现在，我们将转向下一个最重要的向量类型：数值向量。



# 13 数值

## 13.1 引言

数值向量是数据科学的基石，你在本书的前面部分已经多次使用过它们。现在是系统地审视你在 R 中可以对它们做什么的时候了，确保你能够很好地应对任何涉及数值向量的未来问题。

我们将首先为你提供几个工具，以便在你有字符串时生成数字，然后更详细地介绍 `count()` 函数。然后，我们将深入探讨与 `mutate()` 搭配使用的各种数值转换，包括可以应用于其他类型向量的更一般的转换，但通常与数值向量一起使用。最后，我们将介绍与 `summarize()` 搭配使用的汇总函数，并向你展示它们也可以与 `mutate()` 一起使用。

### 13.1.1 必要条件

本章主要使用来自基础 R 的函数，这些函数无需加载任何包即可使用。但我们仍然需要 tidyverse，因为我们将使用 tidyverse 的函数（如 `mutate()` 和 `filter()`）内部使用这些基础 R 函数。和上一章一样，我们将使用 `nycflights13` 数据集的真实示例，以及使用 `c()` 和 `tribble()` 创建的玩具示例。

## 13 数值

```
library(tidyverse)
library(nycflights13)
```

### 13.2 数字化

在大多数情况下，你会获得 R 的数字类型之一的整数或双精度数字。然而，在某些情况下你可能会遇到以字符串形式出现的数字，可能是因为从列标题中转换得到它们，或者是因为在数据导入过程中出现了某些问题。

`readr` 包提供了两个有用的函数来将字符串解析为数字：`parse_double()` 和 `parse_number()`。当你遇到以字符串形式写入的数字时，可以使用 `parse_double()`。

```
x <- c("1.2", "5.6", "1e3")
parse_double(x)
#> [1] 1.2 5.6 1000.0
```

当字符串中包含你想要忽略的非数字文本时，使用 `parse_number()`。这对于货币数据和百分比特别有用：

```
x <- c("$1,234", "USD 3,513", "59%")
parse_number(x)
#> [1] 1234 3513 59
```

## 13.3 计数

令人惊讶的是，仅仅通过计数和一些基本的算术运算你就能够完成不少数据科学工作，因此 `dplyr` 致力于通过 `count()` 函数使计数尽可能简单。这个函数非常适合在分析过程中进行快速探索和检查：

```
flights |> count(dest)
#> # A tibble: 105 x 2
#>   dest     n
#>   <chr> <int>
#> 1 ABQ      254
#> 2 ACK      265
#> 3 ALB      439
#> 4 ANC       8
#> 5 ATL    17215
#> 6 AUS     2439
#> # i 99 more rows
```

尽管在章节 ?? 给出了建议，但我们通常还是将 `count()` 放在单独的一行，因为它通常在控制台中用于快速检查计算是否按预期工作。

如果你想看到最常见的值，添加 `sort = TRUE`:

```
flights |> count(dest, sort = TRUE)
#> # A tibble: 105 x 2
#>   dest     n
#>   <chr> <int>
#> 1 ORD    17283
```

## 13 数值

```
#> 2 ATL    17215  
#> 3 LAX    16174  
#> 4 BOS    15508  
#> 5 MCO    14082  
#> 6 CLT    14064  
#> # i 99 more rows
```

请记住，如果希望查看所有值，可以使用 `|> View()` 或 `|> print(n = Inf)`。

你可以通过 `group_by()`、`summarize()` 和 `n()`“手动”执行相同的计算。这是有用的，因为它允许你同时计算其他汇总统计量：

```
flights |>  
  group_by(dest) |>  
  summarize(  
    n = n(),  
    delay = mean(arr_delay, na.rm = TRUE)  
  )  
#> # A tibble: 105 x 3  
#>   dest      n delay  
#>   <chr> <int> <dbl>  
#> 1 ABQ      254  4.38  
#> 2 ACK      265  4.85  
#> 3 ALB      439 14.4  
#> 4 ANC       8 -2.5  
#> 5 ATL     17215 11.3
```

```
#> 6 AUS    2439  6.02
#> # i 99 more rows
```

`n()` 是一个特殊的汇总函数，它不接受任何参数，而是访问关于“当前”组的信息。这意味着它只能在 `dplyr` 的函数内部使用：

```
n()
#> Error in `n()`:
#> ! Must only be used inside data-masking verbs like `mutate()`,
#> `filter()`, and `group_by()`.
```

`n()` 和 `count()` 有几个变体，你可能会觉得它们有用：

- `n_distinct(x)` 计算一个或多个变量的不同（唯一）值的数量。例如，我们可以找出哪些目的地有最多的航空公司提供服务：

```
flights |>
  group_by(dest) |>
  summarize(carriers = n_distinct(carrier)) |>
  arrange(desc(carriers))

#> # A tibble: 105 x 2
#>   dest   carriers
#>   <chr>     <int>
#> 1 ATL        7
#> 2 BOS        7
#> 3 CLT        7
#> 4 ORD        7
#> 5 TPA        7
```

## 13 数值

```
#> 6 AUS          6  
#> # i 99 more rows
```

- 加权计数是求和。例如你可以“计数”每架飞机飞行的英里数：

```
flights |>  
  group_by(tailnum) |>  
  summarize(miles = sum(distance))  
#> # A tibble: 4,044 x 2  
#>   tailnum    miles  
#>   <chr>     <dbl>  
#> 1 D942DN    3418  
#> 2 NOEGMQ    250866  
#> 3 N10156    115966  
#> 4 N102UW    25722  
#> 5 N103US    24619  
#> 6 N104UW    25157  
#> # i 4,038 more rows
```

加权计数是一个常见的问题，所以 `count()` 函数有一个 `wt` 参数来实现同样的功能：

```
flights |> count(tailnum, wt = distance)
```

- 可以通过结合 `sum()` 和 `is.na()` 来计数缺失值。在 `flights` 数据集中，这表示被取消的航班：

```
flights |>  
  group_by(dest) |>  
  summarize(n_cancelled = sum(is.na(dep_time)))
```

```
#> # A tibble: 105 x 2
#>   dest    n_cancelled
#>   <chr>     <int>
#> 1 ABQ         0
#> 2 ACK         0
#> 3 ALB        20
#> 4 ANC         0
#> 5 ATL       317
#> 6 AUS        21
#> # i 99 more rows
```

### 13.3.1 练习

1. 如何使用 `count()` 来计算给定变量中缺失值的行数?
2. 将以下对 `count()` 的调用扩展为使用 `group_by()`、`summarize()` 和 `arrange()`:
  1. `flights |> count(dest, sort = TRUE)`
  2. `flights |> count(tailnum, wt = distance)`

## 13.4 数值转换

转换函数与 `mutate()` 配合得很好，因为它们的输出长度与输入相同。绝大多数转换函数已经内置在 R 的基本包中。列出所有函数是不切实际的，所以本节将展示一些最有用的。例如，虽然 R 提供了你可能梦寐以求的所有三角函数，但我们在那里没有列出它们，因为它们在数据科学中很少需要。

### 13.4.1 算术和循环规则

在章节 ??，我们介绍了算术运算 (+, -, \*, /, ^) 的基础知识，并在此之后多次使用了它们。这些函数不需要过多的解释，因为它们执行的是你在小学就学过的运算。但是，我们需要简短地讨论一下循环规则（recycling rules），这些规则决定了当左侧和右侧的长度不同时会发生什么。这对于像 `flights |> mutate(air_time = air_time / 60)` 这样的操作很重要，因为在/的左侧有 336,776 个数字，而右侧只有一个。

R 通过循环或重复较短的向量来处理长度不匹配的情况。如果我们在数据框之外创建一些向量，就能更容易地看到这个操作过程：

```
x <- c(1, 2, 10, 20)
x / 5
#> [1] 0.2 0.4 2.0 4.0
# is shorthand for
x / c(5, 5, 5, 5)
#> [1] 0.2 0.4 2.0 4.0
```

通常，你只想循环单个数字（即长度为 1 的向量），但 R 会循环任何较短的向量。如果较长的向量不是较短的向量的倍数，R 通常会（但不总是）给出警告：

```
x * c(1, 2)
#> [1] 1 4 10 40
x * c(1, 2, 3)
#> Warning in x * c(1, 2, 3): longer object length is not a multiple of shorter
#> object length
#> [1] 1 4 30 20
```

## 13.4 数值转换

这些循环规则也适用于逻辑比较 (`==`, `<`, `<=`, `>`, `>=`, `!=`)，如果你不小心使用了 `==` 而不是`%in%`，并且数据框的行数不巧地匹配了这些规则，可能会导致一个令人惊讶的结果。例如，考虑以下代码，它试图找出 1 月和 2 月的所有航班：

```
flights |>
  filter(month == c(1, 2))
#> # A tibble: 25,977 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      542           540       2     923        850
#> 3  2013     1     1      554           600      -6     812        837
#> 4  2013     1     1      555           600      -5     913        854
#> 5  2013     1     1      557           600      -3     838        846
#> 6  2013     1     1      558           600      -2     849        851
#> # i 25,971 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

代码运行无误，但它没有返回你想要的结果。由于循环规则，它找到了在奇数行出发的 1 月航班和在偶数行出发的 2 月航班。不幸的是，由于 `flights` 的行数是偶数所以没有给出警告。

为了防止这种静默失败的情况，tidyverse 的大多数函数使用了一种更严格的循环形式，即仅循环单个值。不幸的是，这在当前情况下或许多其他情况下并不起作用，因为关键的计算是由基础 R 函数 `==` 执行的，而不是 `filter()`。

### 13.4.2 最小值和最大值

算术函数对变量对进行操作。两个紧密相关的函数是 `pmin()` 和 `pmax()`，当给定两个或更多变量时，它们将返回每行中的最小或最大值：

```
df <- tribble(
  ~x, ~y,
  1, 3,
  5, 2,
  7, NA,
)

df |>
  mutate(
    min = pmin(x, y, na.rm = TRUE),
    max = pmax(x, y, na.rm = TRUE)
  )

#> # A tibble: 3 x 4
#>       x     y   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1     1     3     1     3
#> 2     5     2     2     5
#> 3     7    NA     7     7
```

请注意，这些函数与 `min()` 和 `max()` 这样的汇总函数是不同的，后者接受多个观测值并返回一个单一的值。当发现所有的最小值和所有的最大值都相同时，你可以判断你可能使用了错误的形式。

```
df |>
  mutate(
    min = min(x, y, na.rm = TRUE),
    max = max(x, y, na.rm = TRUE)
  )
#> # A tibble: 3 x 4
#>   x     y     min     max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1     1     3     1     7
#> 2     5     2     1     7
#> 3     7    NA     1     7
```

### 13.4.3 模运算

模运算 (modular arithmetic) 是你在学习十进制之前所做的数学运算的技术名称，即除法产生一个整数商和一个余数。在 R 中，`%/%` 执行整数除法，而`%%` 计算余数。

```
1:10 %/%
#> [1] 0 0 1 1 1 2 2 2 3 3
1:10 %% 3
#> [1] 1 2 0 1 2 0 1 2 0 1
```

模运算对于 `flights` 数据集来说很有用，因为我们可以使用它来将 `sched_dep_time` 变量分解为 `hour` 和 `minute`。

## 13 数值

```
flights |>
  mutate(
    hour = sched_dep_time %/% 100,
    minute = sched_dep_time %% 100,
    .keep = "used"
  )
#> # A tibble: 336,776 x 3
#>   sched_dep_time  hour minute
#>   <int>      <dbl>   <dbl>
#> 1          515      5     15
#> 2          529      5     29
#> 3          540      5     40
#> 4          545      5     45
#> 5          600      6     0
#> 6          558      5     58
#> # i 336,770 more rows
```

我们可以将上述方法与来自小节 ?? 的 `mean(is.na(x))` 技巧结合起来查看取消航班的比例在一天中的变化情况。结果如图图 ?? 所示。

```
flights |>
  group_by(hour = sched_dep_time %/% 100) |>
  summarize(prop_cancelled = mean(is.na(dep_time)), n = n()) |>
  filter(hour > 1) |>
  ggplot(aes(x = hour, y = prop_cancelled)) +
  geom_line(color = "grey50") +
  geom_point(aes(size = n))
```

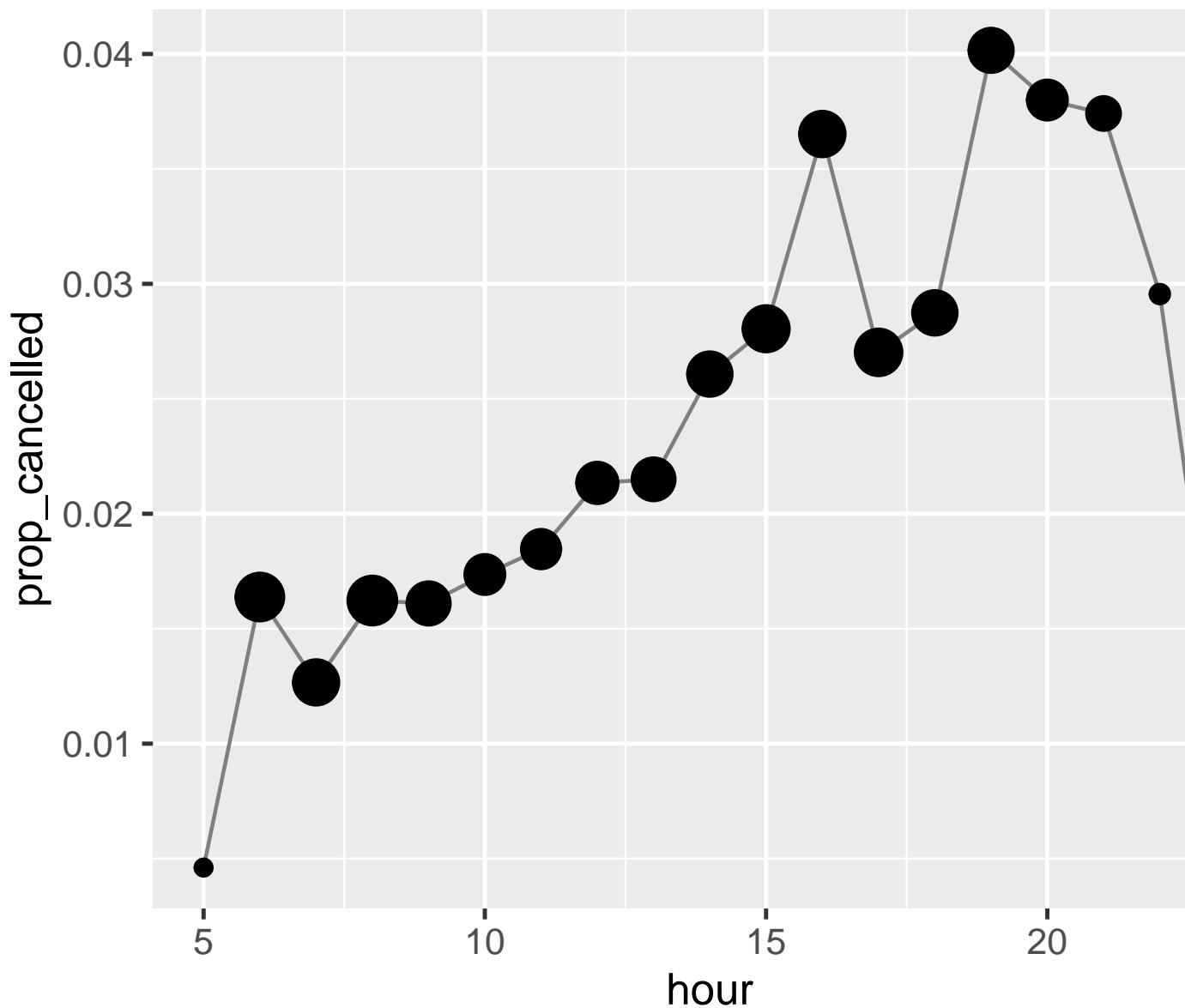


图 13.1: A line plot with scheduled departure hour on the x-axis, and proportion of cancelled flights on the y-axis. Cancellations seem to accumulate over the course of the day until 8pm, very late flights are much less likely to be cancelled.  
445

### 13.4.4 对数

对数在处理跨越多个数量级的数据以及将指数增长转换为线性增长时是非常有用的转换方法。在 R 中，你可以选择三种对数：`log()`（自然对数，底为 e）、`log2()`（底为 2）和 `log10()`（底为 10）。我们推荐使用 `log2()` 或 `log10()`。`log2()` 易于解释，因为在对数尺度上相差 1 对应于原始尺度上的加倍，相差-1 对应于减半；而 `log10()` 易于反向转换，因为（例如）3 是 10 的 3 次方，即 1000。`log()` 的反函数是 `exp()`；要计算 `log2()` 或 `log10()` 的反函数，你需要使用 `2^` 或 `10^`。

### 13.4.5 四舍五入

使用 `round(x)` 将一个数字四舍五入到最接近的整数：

```
round(123.456)
#> [1] 123
```

你可以使用第二个参数 `digits` 来控制四舍五入的精度。`round(x, digits)` 会将 `x` 四舍五入到最近的  $10^{-n}$ ，所以 `digits = 2` 会将 `x` 四舍五入到最近的 0.01。这个定义很有用，因为它意味着 `round(x, -3)` 会将 `x` 四舍五入到最近的千位，而它确实是这样做的：

```
round(123.456, 2) # two digits
#> [1] 123.46
round(123.456, 1) # one digit
#> [1] 123.5
round(123.456, -1) # round to nearest ten
#> [1] 120
```

## 13.4 数值转换

```
round(123.456, -2) # round to nearest hundred  
#> [1] 100
```

`round()` 有一个看起来有些奇怪的特性，第一眼看上去这可能会让人吃惊：

```
round(c(1.5, 2.5))  
#> [1] 2 2
```

`round()` 使用的是所谓的“向偶数舍入”或“银行家舍入”方法：如果一个数字正好在两个整数之间，那么它会被舍入到偶数整数。这是一个很好的策略，因为它使得舍入是无偏的：一半的 0.5 被向上舍入，而另一半被向下舍入。

`round()` 与 `floor()` 和 `ceiling()` 相对应。`floor()` 总是向下舍入，而 `ceiling()` 总是向上舍入。

```
x <- 123.456  
  
floor(x)  
#> [1] 123  
ceiling(x)  
#> [1] 124
```

这些函数 (`floor()` 和 `ceiling()`) 没有 `digits` 参数，所以你可以先将数字缩小比例，然后四舍五入，最后再扩大回原来的比例：

## 13 数值

```
# Round down to nearest two digits
floor(x / 0.01) * 0.01
#> [1] 123.45
# Round up to nearest two digits
ceiling(x / 0.01) * 0.01
#> [1] 123.46
```

如果你想要将 `round()` 舍入到某个数的倍数，你可以使用相同的技巧：

```
# Round to nearest multiple of 4
round(x / 4) * 4
#> [1] 124

# Round to nearest 0.25
round(x / 0.25) * 0.25
#> [1] 123.5
```

### 13.4.6 将数字切分成区间

使用 `cut()`<sup>1</sup> 函数可以将一个数值向量拆分（也称为分箱或分组）成离散的区间：

```
x <- c(1, 2, 5, 10, 15, 20)
cut(x, breaks = c(0, 5, 10, 15, 20))
```

<sup>1</sup>`ggplot2` 提供了一些辅助函数来处理 `cut_interval()`,`cut_number()` 和 `cut_width()` 中的常见情况。虽然 `ggplot2` 是这些函数存放的一个稍显奇怪的地方，但它们作为直方图计算的一部分非常有用，并且在 `tidyverse` 的其他部分出现之前就已经被编写出来了。

## 13.4 数值转换

```
#> [1] (0,5]  (0,5]  (0,5]  (5,10]  (10,15]  (15,20]
#> Levels: (0,5] (5,10] (10,15] (15,20]
```

`breaks` 不需要是等距的：

```
cut(x, breaks = c(0, 5, 10, 100))
#> [1] (0,5]  (0,5]  (0,5]  (5,10]  (10,100] (10,100]
#> Levels: (0,5] (5,10] (10,100]
```

你可以选择性地提供你自己的 `labels`。请注意，`labels` 的数量应该比 `breaks` 少一个。

```
cut(x,
  breaks = c(0, 5, 10, 15, 20),
  labels = c("sm", "md", "lg", "xl")
)
#> [1] sm sm sm md lg xl
#> Levels: sm md lg xl
```

任何超出 `breaks` 范围的值都会变成 `NA`：

```
y <- c(NA, -10, 5, 10, 30)
cut(y, breaks = c(0, 5, 10, 15, 20))
#> [1] <NA>  <NA>  (0,5]  (5,10] <NA>
#> Levels: (0,5] (5,10] (10,15] (15,20]
```

查看文档以了解其他有用的参数，如 `right` 和 `include.lowest`，这些参数控制区间是 `[a, b)` 还是 `(a, b]`，以及是否应将最低区间设为 `[a, b]`。

### 13.4.7 累积和滚动聚合

基础 R 提供了 `cumsum()`,`cumprod()`,`cummin()`,`cummax()` 函数，用于计算连续或累积的和、积、最小值和最大值。dplyr 包提供了 `cummean()` 函数用于计算累积平均值。在实践中，累积和是最常遇到的：

```
x <- 1:10
cumsum(x)
#> [1] 1 3 6 10 15 21 28 36 45 55
```

如果你需要更复杂的滚动或滑动聚合，可以尝试使用[slider](#)包。

### 13.4.8 练习

1. 用文字解释用于生成 @fig-prop-cancelled 的每一行代码的作用。
2. R 提供了什么三角函数？猜测一些名字并查找文档。这些函数用度数还是弧度？
3. 目前，`dep_time` 和 `sched_dep_time` 看起来很方便，但是难以用于计算，因为它们并不是真正的连续数字。你可以通过运行下面的代码来看到基本问题：每小时之间都存在间隙。

```
flights |>
  filter(month == 1, day == 1) |>
  ggplot(aes(x = sched_dep_time, y = dep_delay)) +
  geom_point()
```

将它们转换为更真实的时间表示（可以是自午夜起的小时数（以小数形式）或分钟数）。

4. 将 `dep_time` 和 `arr_time` 四舍五入到最近的五分钟。

## 13.5 通用转换

以下部分描述了一些通常与数值向量一起使用的通用转换，但它们也可以应用于所有其他列类型。

### 13.5.1 秩 (Ranks)

dplyr 提供了一系列受 SQL 启发的排秩函数，但你应该始终从 `dplyr::min_rank()` 开始。它使用了处理并列名次（相持）的标准方法，例如，第 1 名、第 2 名、第 2 名、第 4 名。

```
x <- c(1, 2, 2, 3, 4, NA)
min_rank(x)
#> [1] 1 2 2 4 5 NA
```

请注意，最小的值获得最小的秩；利用 `desc(x)` 可以让最大的值获得最小的秩：

```
min_rank(desc(x))
#> [1] 5 3 3 2 1 NA
```

如果 `min_rank()` 不符合你的需求，请查看它的变体函数 `dplyr::row_number()`、`dplyr::dense_rank()`、`dplyr::percent_rank()` 和 `dplyr::cume_dist()`。查看文档以获取详细信息。

## 13 数值

```
df <- tibble(x = x)
df |>
  mutate(
    row_number = row_number(x),
    dense_rank = dense_rank(x),
    percent_rank = percent_rank(x),
    cume_dist = cume_dist(x)
  )
#> # A tibble: 6 x 5
#>   x     row_number  dense_rank  percent_rank  cume_dist
#>   <dbl>      <int>       <int>        <dbl>       <dbl>
#> 1     1          1           1         0.000     0.2
#> 2     2          2           2         0.25      0.6
#> 3     2          3           2         0.25      0.6
#> 4     3          4           3         0.75      0.8
#> 5     4          5           4         1.00      1.0
#> 6     NA         NA          NA         NA         NA
```

你可以通过选择适当的 `ties.method` 参数来使用 R 的基础函数 `rank()` 来达到许多相同的结果；你也可能希望通过设置 `na.last = "keep"` 将 NAs 值保留为 NAs。

`row_number()` 在 dplyr 的函数内部使用时也可以不带任何参数。在这种情况下它会给出“当前”行的编号。当与`%%` 或`%/%` 结合使用时，它可以成为将数据划分为大小相似的组的有用工具：

```
df <- tibble(id = 1:10)
```

```
df |>
  mutate(
    row0 = row_number() - 1,
    three_groups = row0 %% 3,
    three_in_each_group = row0 %/% 3
  )
#> # A tibble: 10 x 4
#>   id   row0 three_groups three_in_each_group
#>   <int> <dbl>        <dbl>                <dbl>
#> 1     1     0           0                  0
#> 2     2     1           1                  0
#> 3     3     2           2                  0
#> 4     4     3           0                  1
#> 5     5     4           1                  1
#> 6     6     5           2                  1
#> # i 4 more rows
```

### 13.5.2 偏移量 (Offsets)

`dplyr::lead()` 和 `dplyr::lag()` 允许你引用“当前”值之前或之后的值。它们会返回一个与输入长度相同的向量，并在开始或结束时用 NAs 填充：

```
x <- c(2, 5, 11, 11, 19, 35)
lag(x)
#> [1] NA  2  5 11 11 19
lead(x)
#> [1]  5 11 11 19 35 NA
```

## 13 数值

- `x - lag(x)` 给出当前值和前一个值之间的差值。

```
x - lag(x)
#> [1] NA 3 6 0 8 16
```

- `x == lag(x)` 告诉你当前值何时发生改变。

```
x == lag(x)
#> [1] NA FALSE FALSE TRUE FALSE FALSE
```

你可以通过使用第二个参数 `n` 来实现超过一个位置的向前或向后取值。

### 13.5.3 连续标识符

有时，每当某个事件发生时你都希望开始一个新组。例如在查看网站数据时，你通常会希望将事件划分为不同会话（sessions），即当距离上次活动超过 `x` 分钟时你会开始一个新会话。比如，想象你有某人访问网站的时间记录：

```
events <- tibble(
  time = c(0, 1, 2, 3, 5, 10, 12, 15, 17, 19, 20, 27, 28, 30)
)
```

并且你已经计算了每个事件之间的时间间隔，并判断了是否存在足够大的间隔以符合标准：

```
events <- events |>
  mutate(
    diff = time - lag(time, default = first(time)),
    has_gap = diff >= 5
)
```

```
)
events
#> # A tibble: 14 x 3
#>   time    diff has_gap
#>   <dbl> <dbl> <lgl>
#> 1     0     0 FALSE
#> 2     1     1 FALSE
#> 3     2     1 FALSE
#> 4     3     1 FALSE
#> 5     5     2 FALSE
#> 6    10     5 TRUE
#> # i 8 more rows
```

但是，我们如何从逻辑向量转换为可以使用 `group_by()` 进行分组的东西呢?`cumsum()` 函数(来自小节 ??)此时就派上了用场,当存在间隔(即 `has_gap` 为 `TRUE`)时,它会增加分组编号(来自 @sec-numeric-summaries-of-logicals):

```
events |> mutate(
  group = cumsum(has_gap)
)
#> # A tibble: 14 x 4
#>   time    diff has_gap group
#>   <dbl> <dbl> <lgl>   <int>
#> 1     0     0 FALSE     0
#> 2     1     1 FALSE     0
#> 3     2     1 FALSE     0
#> 4     3     1 FALSE     0
#> 5     5     2 FALSE     0
```

## 13 数值

```
#> 6      10      5 TRUE      1  
#> # i 8 more rows
```

创建分组变量的另一种方法是 `consecutive_id()`，它会在其参数之一发生变化时开始一个新组。例如，受到[this stackoverflow question](#)的启发，假设你有一个包含许多重复值的数据框：

```
df <- tibble(  
  x = c("a", "a", "a", "b", "c", "c", "d", "e", "a", "a", "b", "b"),  
  y = c(1, 2, 3, 2, 4, 1, 3, 9, 4, 8, 10, 199)  
)
```

如果你想要保留每个重复值 `x` 的第一行，你可以使用 `group_by()`、`consecutive_id()` 和 `slice_head()`。

```
df |>  
  group_by(id = consecutive_id(x)) |>  
  slice_head(n = 1)  
#> # A tibble: 7 x 3  
#> # Groups:   id [7]  
#>   x       y     id  
#>   <chr> <dbl> <int>  
#> 1 a       1     1  
#> 2 b       2     2  
#> 3 c       4     3  
#> 4 d       3     4  
#> 5 e       9     5
```

```
#> 6 a      4      6
#> # i 1 more row
```

### 13.5.4 练习

1. 使用排秩函数找出延误时间最长的 10 个航班。如果出现并列的情况，你打算如何处理？请仔细阅读 `min_rank()` 函数的文档。
2. 哪架飞机（`tailnum`）的准时记录最差？
3. 如果你想尽可能避免延误，你应该在一天中的哪个时间段飞行？
4. `flights |> group_by(dest) |> filter(row_number() < 4)` 这段代码做了什么？`flights |> group_by(dest) |> filter(row_number(dep_delay) < 4)` 这段代码又做了什么？
5. 对于每个目的地，计算延误的总分钟数；对于每个航班，计算其目的地延误总时间中所占的比例。
6. 延误通常是时间相关的：即使导致初始延误的问题已经解决，后续的航班仍然会延误以允许前面的航班起飞。使用 `lag()` 函数，探索某小时的平均航班延误与前一小时的平均延误之间的关系。

```
flights |>
  mutate(hour = dep_time %/% 100) |>
  group_by(year, month, day, hour) |>
  summarize(
    dep_delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .groups = "drop"
```

```
) |>
filter(n > 5)
```

7. 查看每个目的地，你能找到那些可疑的快速航班（即可能存在数据录入错误的航班）吗？计算某航班的空中飞行时间与该目的地最短飞行时间的相对值。哪些航班在空中延误最严重？
8. 找出至少由两家航空公司运营的所有目的地。使用这些目的地，根据同目的地的表现，对航空公司进行相对排名。

## 13.6 数值汇总

仅使用我们已经介绍过的 `counts`、`means` 和 `sums` 就可以让你走得很远，但 R 提供了许多其他有用的汇总函数。以下是可能会有用的一些函数。

### 13.6.1 中心

到目前为止，我们主要使用 `mean()` 来汇总数值向量的中心。正如在 @sec-sample-size 看到的，由于均数是总和除以总数得到的，因此即使对几个异常高或异常低的值它也很敏感。一种替代方法是使用 `median()`，它会找到一个位于向量“中间”的值，即 50% 的值高于它，50% 的值低于它。根据你感兴趣的变量的分布形状，选取均数或者中位数作为合适的中心度量标准。例如对于对称分布我们通常报告均数，而对于偏态分布则报告中位数。

图 ?? 比较了每个目的地的起飞延误均数与中位数（以分钟为单位）。延误中位数总是小于延误均数，因为航班有时会延误数小时，但永远不会提前数小时起飞。

## 13.6 数值汇总

```
flights |>
  group_by(year, month, day) |>
  summarize(
    mean = mean(dep_delay, na.rm = TRUE),
    median = median(dep_delay, na.rm = TRUE),
    n = n(),
    .groups = "drop"
  ) |>
  ggplot(aes(x = mean, y = median)) +
  geom_abline(slope = 1, intercept = 0, color = "white", linewidth = 2) +
  geom_point()
```

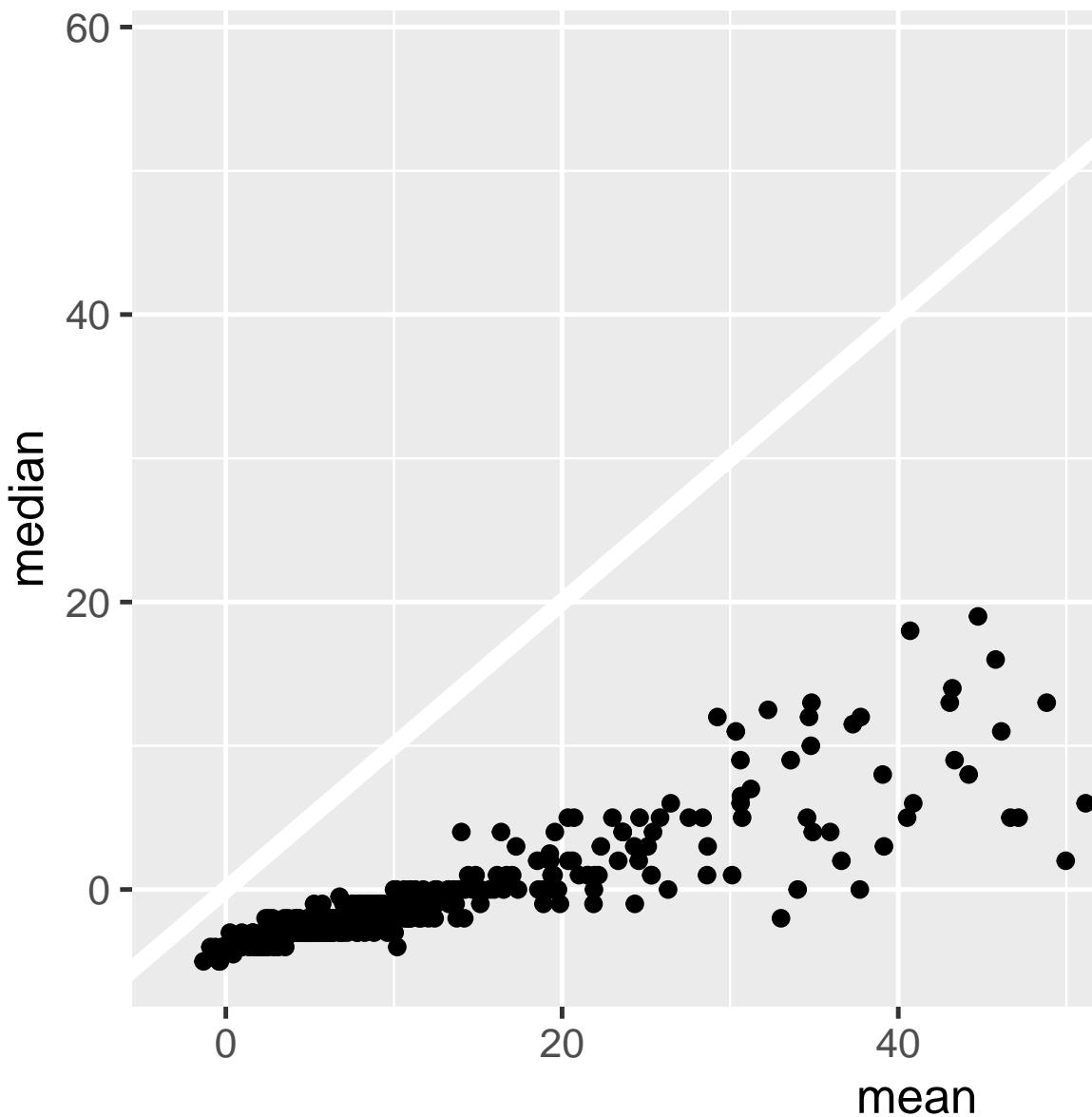


图 13.2: A scatterplot showing the differences of summarizing daily departure delay with median instead of mean.

你可能还想知道众数 (mode)，也就是最常见的值。这是一种仅在非常简单的情况下才有效的汇总统计量（这就是为什么你可能在高中学过它），但它在许多真实数据集上并不适用。如果数据是离散的，可能存在多个最常见的值，而如果数据是连续的，则可能没有最常见的值，因为每个值都略有不同。由于这些原因，统计学家往往不使用众数，并且在 R 的基础包中也没有包含计算众数<sup>2</sup>的函数。

### 13.6.2 最小值、最大值和分位数

如果你对中心位置以外的其他位置感兴趣怎么办？`min()` 和 `max()` 会给你最大值和最小值。另一个强大的工具是 `quantile()`，它是中位数的泛化：`quantile(x, 0.25)` 会找到大于 25% 的 x 的值，`quantile(x, 0.5)` 相当于中位数，而 `quantile(x, 0.95)` 会找到大于 95% 的 x 的值。

对于 `flights` 数据，你可能想查看延误的 95% 分位数而不是最大值，因为它会忽略延误最严重的 5% 航班，这部分航班可能非常极端。

```
flights |>
  group_by(year, month, day) |>
  summarize(
    max = max(dep_delay, na.rm = TRUE),
    q95 = quantile(dep_delay, 0.95, na.rm = TRUE),
    .groups = "drop"
  )
#> # A tibble: 365 x 5
#>   year month   day   max   q95
#>   <int> <int> <int> <dbl> <dbl>
```

---

<sup>2</sup> `mode()` 函数做的是完全不同的事情！

## 13 数值

```
#> 1 2013 1 1 853 70.1
#> 2 2013 1 2 379 85
#> 3 2013 1 3 291 68
#> 4 2013 1 4 288 60
#> 5 2013 1 5 327 41
#> 6 2013 1 6 202 51
#> # i 359 more rows
```

### 13.6.3 离散性 (Spread)

有时你并不是那么关心大部分数据所在的位置，而是关心它的离散程度。两个常用的汇总统计量是标准差 `sd(x)` 和四分位距 `IQR()`。我们在这里不会解释 `sd()`，因为你可能已经很熟悉了，但 `IQR()` 可能比较新颖，它等于 `quantile(x, 0.75) - quantile(x, 0.25)`，给出了包含中间 50% 数据的范围。

我们可以利用这个方法来揭示 `flights` 数据中的一个小异常。你可能会认为，由于机场总是在固定的位置，所以起点和终点之间距离的离散性应该是零。但是下面的代码揭示了机场 [EGE](#) 的一个数据异常：

```
flights |>
  group_by(origin, dest) |>
  summarize(
    distance_iqr = IQR(distance),
    n = n(),
    .groups = "drop"
  ) |>
  filter(distance_iqr > 0)
```

```
#> # A tibble: 2 x 4
#>   origin dest  distance_iqr     n
#>   <chr>   <chr>      <dbl> <int>
#> 1 EWR     EGE        1     110
#> 2 JFK     EGE        1     103
```

### 13.6.4 分布

值得记住的是，上述所有描述的汇总统计量都是将分布简化为单个数字的一种方式。这意味着它们本质上是简化的，如果你选择了错误的汇总方式，很容易忽略各组之间的重要差异。因此，在确定汇总统计量之前，先对数据进行可视化始终是个好主意。

图 ?? 显示了出发延误的整体分布。这个分布是如此偏斜，以至于我们必须放大才能看到大部分数据。这表明均数可能不是一个好的汇总方式，我们可能更喜欢使用中位数。

同样，检查子组的分布是否与整体相似也是一个好主意。在下面的图中，`dep_delay` 的 365 个频数多边形图（frequency polygons）被叠加在一起，每天一个。这些分布似乎遵循一个共同的模式，这表明每天使用相同的汇总方式是合适的。

```
flights |>
  filter(dep_delay < 120) |>
  ggplot(aes(x = dep_delay, group = interaction(day, month))) +
  geom_freqpoly(binwidth = 5, alpha = 1/5)
```

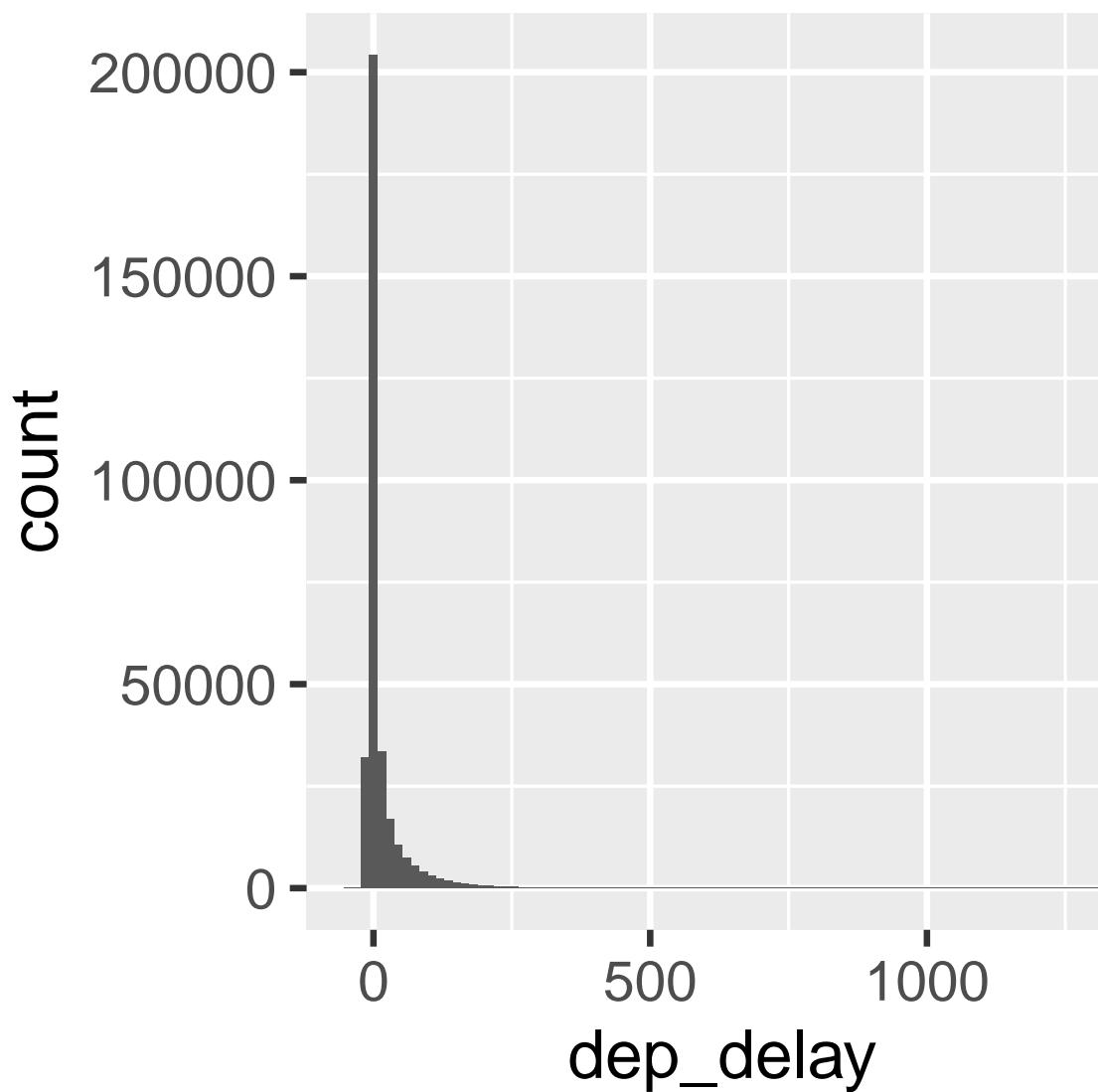
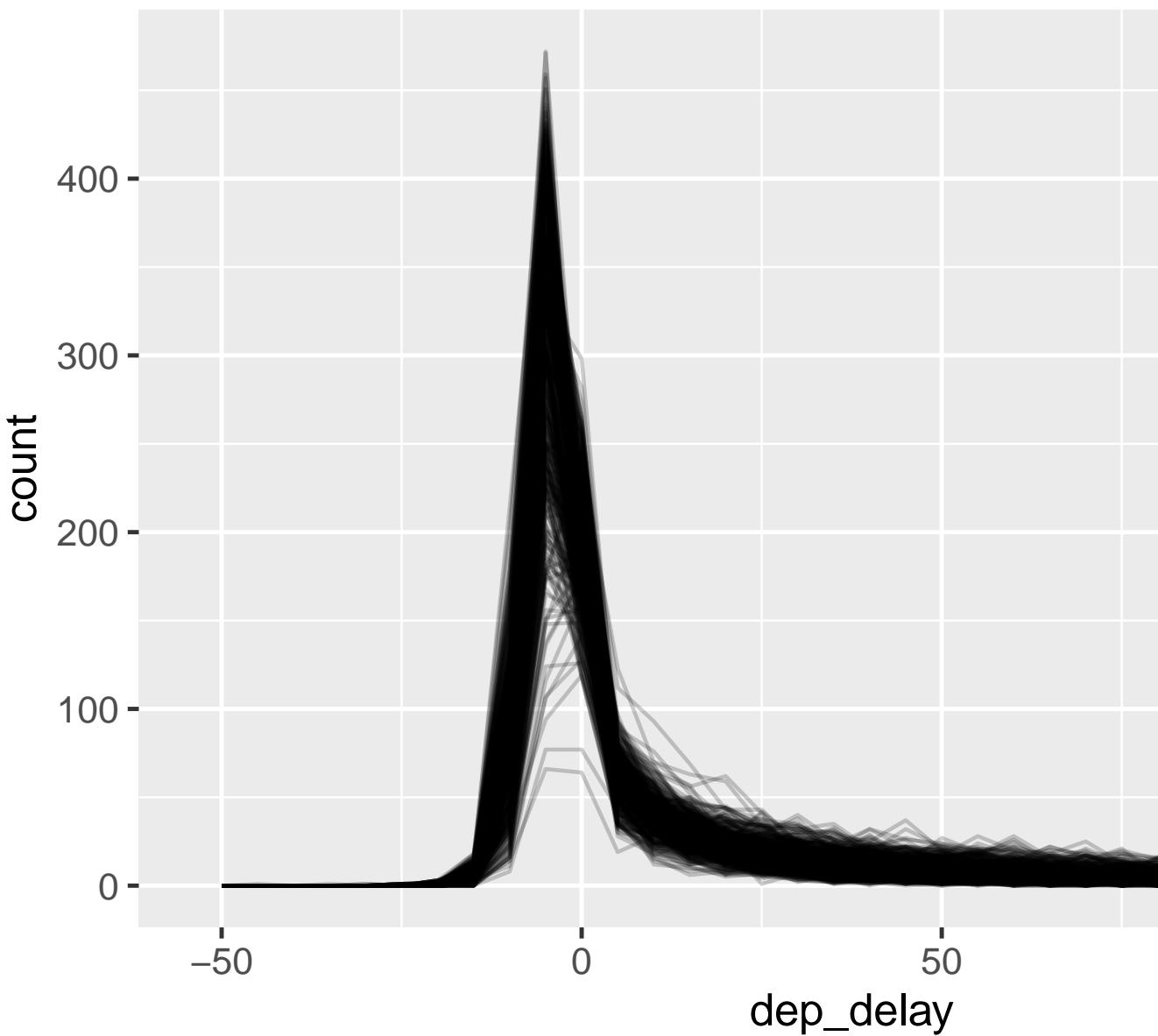


图 13.3: (Left) The histogram of the full data is extremely skewed making it hard to get any details. (Right) Zooming into delays of less than two hours makes it possible to see what's happening with the bulk of the observations.

13.6 数值汇总



465

## 13 数值

不要害怕探讨针对你正在处理的数据自定义的汇总方法。在这种情况下，这可能意味着分别汇总提前起飞的航班和晚起飞的航班的分布，或者鉴于这些值严重偏斜，你可能还需尝试进行对数转换。最后，不要忘记在 @sec-sample-size 中学到的内容：每当创建数值汇总时，最好包括每个组的观测数。

### 13.6.5 位置

对于数值向量，还有最后一种类型的汇总统计量非常有用，它也适用于其他类型的值：提取特定位置的值：`first(x)`、`last(x)` 和 `nth(x, n)`。

例如，我们可以找到每天第一次、第五次和最后一次的出发时间：

```
flights |>
  group_by(year, month, day) |>
  summarize(
    first_dep = first(dep_time, na_rm = TRUE),
    fifth_dep = nth(dep_time, 5, na_rm = TRUE),
    last_dep = last(dep_time, na_rm = TRUE)
  )
#> `summarise()` has grouped output by 'year', 'month'. You can override using
#> the ` `.groups` argument.
#> # A tibble: 365 x 6
#> # Groups:   year, month [12]
#>   year month   day first_dep fifth_dep last_dep
#>   <int> <int> <int>     <int>      <int>     <int>
#> 1  2013     1     1       517       554     2356
#> 2  2013     1     2       42        535     2354
#> 3  2013     1     3       32        520     2349
```

## 13.6 数值汇总

```
#> 4 2013 1 4 25 531 2358  
#> 5 2013 1 5 14 534 2357  
#> 6 2013 1 6 16 555 2355  
#> # i 359 more rows
```

(注意：因为 dplyr 函数使用 `_` 来分隔函数名和参数名的组件，所以这些函数使用 `na_rm` 而不是 `na.rm`)

如果你熟悉 `[` 操作符，我们将在 @sec-subset-many 中再次讨论它，你可能会想是否需要这些函数。有三个原因：默认参数允许你在指定的位置不存在时提供一个默认值，`order_by` 参数允许你局部覆盖行的顺序，而 `na_rm` 参数允许你删除缺失值。

按位置提取值是对按秩筛选的补充。筛选操作会返回所有变量，每个观测都在单独的一行中：

```
flights |>  
  group_by(year, month, day) |>  
  mutate(r = min_rank(sched_dep_time)) |>  
  filter(r %in% c(1, max(r)))  
#> # A tibble: 1,195 x 20  
#> # Groups:   year, month, day [365]  
#>   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     2353         2359       -6     425          445  
#> 3 2013     1     1     2353         2359       -6     418          442  
#> 4 2013     1     1     2356         2359       -3     425          437  
#> 5 2013     1     2      42          2359       43     518          442
```

## 13 数值

```
#> 6 2013 1 2 458 500 -2 703 650  
#> # i 1,189 more rows  
#> # i 12 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, ...
```

### 13.6.6 利用 `mutate()`

顾名思义，汇总函数通常与 `summarize()` 一起使用。但是，因为我们在 @sec-recycling 中讨论的循环规则，它们也可以与 `mutate()` 一起使用，特别是当你想进行某种分组标准化时。例如：

- `x / sum(x)` 计算了 `x` 中每个元素占总和的比例；
- `(x - mean(x)) / sd(x)` 计算 Z-score（将 `x` 标准化为均数为 0，标准差为 1）；
- `(x - min(x)) / (max(x) - min(x))` 将 `x` 标准化到范围 [0, 1]；
- `x / first(x)` 基于第一个观测值计算一个指数。

### 13.6.7 练习

1. 头脑风暴至少 5 种不同的方法来评估一组航班的典型延误特性。什么时候用 `mean()`? 什么时候用 `median()`? 什么情况下你可能想使用其他方法? 你应该使用到达延误还是起飞延误? 为什么你可能想使用来自 `planes` 的数据?
2. 哪些目的地的航速变化最大?
3. 创建一个图来进一步探索 EGE 的冒险经历。你能找到机场变换位置的任何证据吗? 你能找到可解释这种差异的另一个变量吗?

## 13.7 小结

你已经熟悉了许多用于数字处理的工具，在阅读本章之后也知道了如何在 R 中使用它们。你还学习了一些有用的通用转换方法，这些方法通常（但不仅限于）应用于数字向量，如秩和偏移量。最后，通过一些数字汇总进行了实践，并讨论了几个你应该考虑的统计学挑战。

在接下来的两章中，我们将使用 `stringr` 包深入研究字符串的处理。字符串是一个很大的主题，因此它们被分为两章，一章关于字符串的基础知识，另一章关于正则表达式。



# 14 字符串

## 14.1 引言

到目前为止，你已经使用了一堆字符串，但对它们的细节了解并不多，现在是时候深入了解它们了。学习字符串的工作原理，并掌握一些可以使用的字符串操作工具。

我们将从创建字符串和字符向量的细节开始。然后，你将深入了解如何从数据中创建字符串，然后是从数据中提取字符串。接着，我们将讨论处理单个字母的工具。本章最后介绍了一些处理单个字母的函数，并简要讨论了在使用其他语言时，你对英语的预期可能会引导你误入歧途的情况。

在下一章中，我们将继续使用字符串，届时你将深入了解正则表达式的强大功能。

### 14.1.1 必要条件

在本章中，我们将使用 `stringr` 包中的函数，`stringr` 是 tidyverse 的一部分。我们还将使用 `babynames` 数据，因为它提供了一些有趣的字符串供操作。

```
library(tidyverse)
library(babynames)
```

当你使用 `stringr` 函数时，你可以很容易地识别出来，因为所有的 `stringr` 函数都以 `str_` 开头。如果你使用 RStudio，这尤其有用，因为输入 `str_` 会触发自动补全功能，帮助你回忆起可用的函数。

The screenshot shows the RStudio code editor with the cursor at `str_`. A completion dropdown menu is open, listing several functions starting with `str_`, such as `str_c`, `str_conv`, `str_count`, `str_detect`, `str_dup`, `str_extract`, and `str_extract_all`. A tooltip provides a detailed explanation of how `str_c` works when building up a matrix of strings. It states that each input argument must have a length equal to or expandable to the length of the longest argument, using the usual recycling rules. The `sep` string is inserted between each column. If `collapse` is `NULL`, each row is collapsed into a single string. If `collapse` is a string, it is inserted at the end of each row, and the entire matrix collapsed to a single string.

## 14.2 生成一个字符串

在本书的前面部分，我们曾经创建过字符串，但没有讨论细节。首先，你可以使用单引号（'）或双引号（"）来创建字符串。两者在行为上没有区别，因此，为了保持一致性，[tidyverse 风格指南](#) 建议使用双引号（"），除非字符串中包含多个"。

```
string1 <- "This is a string"
string2 <- 'If I want to include a "quote" inside a string, I use single quotes'
```

如果你漏写了结束引号，你会看到 +，这是继续提示符：

```
> "This is a string without a closing quote
+
+
+ HELP I'M STUCK IN A STRING
```

## 14.2 生成一个字符串

如果碰到了这种情况，而你又不知道漏写了哪个引号，可以按 Esc 键取消并重新尝试。

### 14.2.1 转义符 (Escapes)

要在字符串中包含单引号或双引号，您可以使用\ 来“转义”它：

```
double_quote <- "\\" # or '\"'  
single_quote <- '\\' # or '\"'
```

所以如果你想在字符串中包含一个反斜杠，你需要转义它:"\\":

```
backslash <- "\\\"
```

请注意，字符串的输出形式与字符串本身并不相同，因为输出形式会显示转义符（换句话说，当你输出一个字符串时，你可以复制并粘贴输出来重新创建该字符串）。要查看字符串的原始内容，请使用 `str_view()`<sup>1</sup>：

```
x <- c(single_quote, double_quote, backslash)  
x  
#> [1] '\"' "\\" "\\\\"  
  
str_view(x)  
#> [1] | '  
#> [2] | '\"'  
#> [3] | '\\'
```

---

<sup>1</sup>或使用基础 R 函数 `writeLines()`.

### 14.2.2 原始字符串

使用多个引号或反斜杠创建字符串很快就会让人困惑。为了说明这个问题，我们创建一个字符串，其中包含定义 `double_quote` 和 `single_quote` 变量的代码块的内容：

```
tricky <- "double_quote <- \"\\\\\" # or '\\"'
single_quote <- '\\\' # or '\"'
str_view(tricky)
#> [1] | double_quote <- \"\" # or ''
#>       | single_quote <- '\\\' # or ""
```

这么多反斜杠!(这有时被称为[斜杠综合症](#))为了消除转义，你可以使用原始字符串<sup>2</sup>：

```
tricky <- r"(double_quote <- \"\" # or ''
single_quote <- '\\\' # or '')"
str_view(tricky)
#> [1] | double_quote <- \"\" # or ''
#>       | single_quote <- '\\\' # or ""
```

原始字符串通常以 `r"(开头并以)"` 结尾。但是，如果你的字符串包含"`"`，你可以改用 `r"[]"` 或 `r"{}"`，如果这还不够，你可以插入任意数量的短横线来使开闭对变得唯一，例如 `r"--()--"`、`r"---()---` 等。原始字符串足够灵活，可以处理任何文本。

---

<sup>2</sup>在 R 4.0.0 及更高版本中可用。

### 14.2.3 其他特殊字符

除了\"、\'和\\之外，还有其他一些可能有用的特殊字符。最常见的是\n（换行）和\t（制表符）。有时你还会看到包含以\u或\U开头的 Unicode 转义序列的字符串。这是一种在所有系统上都能工作的非英文字符的写法。你可以在?Quotes中查看其他特殊字符的完整列表。

```
x <- c("one\ntwo", "one\ttwo", "\u00b5", "\u0001f604")
x
#> [1] "one\ntwo"  "one\ttwo"  "\u00b5"      ""
str_view(x)
#> [1] | one
#>     | two
#> [2] | one{\t}two
#> [3] | \u00b5
#> [4] |
```

请注意，`str_view()` 使用花括号来表示制表符，以便更容易地发现它们<sup>3</sup>。处理文本时的一个挑战是文本中可能存在多种不同形式的空白字符，因此了解这种背景知识有助于你识别是否发生了异常情况。

### 14.2.4 练习

1. 创建包含以下值的字符串：

1. He said "That's amazing!"

---

<sup>3</sup>`str_view()` 也使用颜色来引起你对制表符、空格、匹配等的注意。这些颜色目前不会在书中显示，但在交互式运行代码时你会注意到它们。

## 14 字符串

2. \a\b\c\d
  3. \\\\\\\
2. 在 R 会话中创建字符串并输出它。特殊字符 “\u00a0” 发生了什么? `str_view()` 是如何显示它的? 你能在谷歌上查一下这个特殊字符是什么吗?

```
x <- "This\u00a0is\u00a0tricky"
```

## 14.3 从数据中创建许多字符串

既然你已经学会了“手动”创建一两个字符串的基础知识，接下来我们将详细讨论如何从其他字符串中创建字符串。这将帮助你解决一个常见问题，即当你有一些自己写的文本并希望将其与数据框中的字符串结合时。例如，你可能想要将“Hello”与一个名字变量结合起来创建一个问候语。我们将向你展示如何使用 `str_c()` 和 `str_glue()` 函数来完成这个任务，以及如何在 `mutate()` 函数中使用它们。这自然会引出一个问题，即你可能在 `summarize()` 函数中使用哪些 `stringr` 函数，因此我们将以讨论 `str_flatten()` 函数来结束这一节，它是一个用于字符串的汇总函数。

### 14.3.1 `str_c()`

`str_c()` 可以接受任意数量的向量作为参数并返回一个字符向量：

### 14.3 从数据中创建许多字符串

```
str_c("x", "y")
#> [1] "xy"
str_c("x", "y", "z")
#> [1] "xyz"
str_c("Hello ", c("John", "Susan"))
#> [1] "Hello John"  "Hello Susan"
```

`str_c()` 的功能类似于基础 R 的 `paste0()`，但它是为了与 `mutate()` 一起使用而设计的，它遵守 tidyverse 中关于循环（recycling）和缺失值传播（propagating missing values）的惯常规则。

```
df <- tibble(name = c("Flora", "David", "Terra", NA))
df |> mutate(greeting = str_c("Hi ", name, "!"))
#> # A tibble: 4 x 2
#>   name   greeting
#>   <chr>  <chr>
#> 1 Flora Hi Flora!
#> 2 David Hi David!
#> 3 Terra Hi Terra!
#> 4 <NA>   <NA>
```

如果你想以另一种方式显示缺失值，可以使用 `coalesce()` 函数来替换它们。根据你的需求，可以在 `str_c()` 函数内部或外部使用 `coalesce()`。

```
df |>
  mutate(
    greeting1 = str_c("Hi ", coalesce(name, "you"), "!"),
    greeting2 = coalesce(str_c("Hi ", name, "!"), "Hi!")
```

## 14 字符串

```
)  
#> # A tibble: 4 x 3  
#>   name  greeting1 greeting2  
#>   <chr> <chr>     <chr>  
#> 1 Flora Hi Flora! Hi Flora!  
#> 2 David Hi David! Hi David!  
#> 3 Terra Hi Terra! Hi Terra!  
#> 4 <NA>  Hi you!    Hi!
```

### 14.3.2 str\_glue()

如果在使用 `str_c()` 时混合了很多固定的和可变的字符串，你会发现需要输入很多"s，这使得很难看清代码的总体目标。为此，`glue`包提供了一个替代方法，即 `str_glue()`<sup>4</sup>函数。你只需给它一个具有特殊功能的单一字符串：{} 内的任何内容都会像在引号外部一样被评估。

```
df |> mutate(greeting = str_glue("Hi {name}!"))  
#> # A tibble: 4 x 2  
#>   name  greeting  
#>   <chr> <glue>  
#> 1 Flora Hi Flora!  
#> 2 David Hi David!  
#> 3 Terra Hi Terra!  
#> 4 <NA>  Hi NA!
```

---

<sup>4</sup>如果不使用 `stringr`，你也可以用 `glue::glue()` 直接访问它。

## 14.3 从数据中创建许多字符串

正如你所看到的，`str_glue()` 目前将缺失值转换为字符串“NA”，不幸的是这与 `str_c()` 的处理方式不一致。

你也可能会想，如果需要在字符串中包含常规的 {或} 符号时应该怎么办。如果你猜测需要以某种方式转义它，那么你的思路是正确的。窍门在于 `glue` 使用了一种略有不同的转义技术：不是使用像\这样的特殊字符作为前缀，而是将特殊字符重复两次：

```
df |> mutate(greeting = str_glue("{{Hi {name}!}}"))  
#> # A tibble: 4 x 2  
#>   name  greeting  
#>   <chr> <glue>  
#> 1 Flora {{Hi Flora!}}  
#> 2 David {{Hi David!}}  
#> 3 Terra {{Hi Terra!}}  
#> 4 <NA>  {{Hi NA!}}
```

### 14.3.3 `str_flatten()`

`str_c()` 和 `str_glue()` 在与 `mutate()` 一起使用时效果很好，因为它们的输出与输入具有相同的长度。但是，如果你想要一个与 `summarize()` 配合良好的函数，即一个总是返回一个单独字符串的函数，那么 `str_flatten()`<sup>5</sup> 就是为此而设计的：它接受一个字符向量，并将向量的每个元素组合成一个单独的字符串。

---

<sup>5</sup>The base R equivalent is `paste()` used with the `collapse` argument.

## 14 字符串

```
str_flatten(c("x", "y", "z"))
#> [1] "xyz"
str_flatten(c("x", "y", "z"), ", ")
#> [1] "x, y, z"
str_flatten(c("x", "y", "z"), ", ", last = ", and ")
#> [1] "x, y, and z"
```

这使得它可以很好地与 `summarize()` 一起工作：

```
df <- tribble(
  ~ name, ~ fruit,
  "Carmen", "banana",
  "Carmen", "apple",
  "Marvin", "nectarine",
  "Terence", "cantaloupe",
  "Terence", "papaya",
  "Terence", "mandarin"
)
df |>
  group_by(name) |>
  summarize(fruits = str_flatten(fruit, ", "))
#> # A tibble: 3 x 2
#>   name    fruits
#>   <chr>   <chr>
#> 1 Carmen  banana, apple
#> 2 Marvin  nectarine
#> 3 Terence cantaloupe, papaya, mandarin
```

### 14.3.4 练习

- 比较和对比 `paste0()` 和 `str_c()` 对于以下输入的结果:

```
str_c("hi ", NA)
str_c(letters[1:2], letters[1:3])
```

- `paste()` 和 `paste0()` 有什么区别? 如何用 `str_c()` 重建与 `paste()` 等效的功能?
- 将以下表达式从 `str_c()` 转换为 `str_glue()`, 反之亦然:

- `str_c("The price of ", food, " is ", price)`
- `str_glue("I'm {age} years old and live in {country}")`
- `str_c("\\section{", title, "}")`

## 14.4 从字符串提取数据

将多个变量压缩到单个字符串中是很常见的。在本节中, 你将学习如何使用四个 `tidyverse` 函数来提取它们:

- `df |> separate_longer_delim(col, delim)`
- `df |> separate_longer_position(col, width)`
- `df |> separate_wider_delim(col, delim, names)`
- `df |> separate_wider_position(col, widths)`

如果仔细观察, 就会发现这里有一个共同的模式: 首先 `separate_`, 然后 `longer` 或 `wider`, 接着 `_`, 最后通过分隔符或位置来进一步处理。这是因为这四个函数是由两个更简单的原语组成的:

- 正如 `pivot_longer()` 和 `pivot_wider()` 一样，以 `_longer` 结尾的函数通过创建新的行来使输入的数据框变长，而以 `_wider` 结尾的函数则通过生成新的列来使输入的数据框变宽。
- `delim` 使用像 "`,`"、"`"` 或 "`" "` 这样的分隔符来拆分字符串；而 `position` 则按照指定的宽度进行拆分，如 `c(3, 5, 2)`。

我们将在 @sec-regular-expressions 中再次回到这个家族中的最后一个成员 `separate_wider_regex()`。它是这些 `wider` 函数中最灵活的，但你需要对正则表达式有所了解才能使用它。

接下来的两个部分将向你介绍这些拆分函数背后的基本思想，首先是按行拆分（这稍微简单一些），然后是按列拆分。最后，将讨论 `wider` 函数提供的用于诊断问题的工具。

#### 14.4.1 拆分成行

当字符串中组件的数量在每行之间变化时，将字符串拆分成行往往是最有用的。最常见的情况是，需要 `separate_longer_delim()` 基于分隔符进行拆分：

```
df1 <- tibble(x = c("a,b,c", "d,e", "f"))
df1 |>
  separate_longer_delim(x, delim = ",")
#> # A tibble: 6 x 1
#>   x
#>   <chr>
#> 1 a
#> 2 b
#> 3 c
```

```
#> 4 d  
#> 5 e  
#> 6 f
```

在实际情况中，`separate_longer_position()` 的使用较少见，但一些较旧的数据集确实使用了非常紧凑的格式，其中每个字符都用于记录一个值：

```
df2 <- tibble(x = c("1211", "131", "21"))  
df2 |>  
  separate_longer_position(x, width = 1)  
#> # A tibble: 9 x 1  
#>   x  
#>   <chr>  
#> 1 1  
#> 2 2  
#> 3 1  
#> 4 1  
#> 5 1  
#> 6 3  
#> # i 3 more rows
```

### 14.4.2 拆分成列

当每个字符串中组件数量固定，并且希望将它们分散到列中时，将字符串拆分成列往往是最有用的。它们比其 `longer` 对等项稍微复杂一些，因为你需要为列命名。例如，在以下数据集中，`x` 由代码、版号和年份组成，它们之间由“.”

## 14 字符串

分隔。要使用 `separate_wider_delim()`，我们需要在两个参数中提供分隔符和列名：

```
df3 <- tibble(x = c("a10.1.2022", "b10.2.2011", "e15.1.2015"))
df3 |>
  separate_wider_delim(
    x,
    delim = ".",
    names = c("code", "edition", "year")
  )
#> # A tibble: 3 x 3
#>   code  edition year
#>   <chr> <chr>   <chr>
#> 1 a10    1      2022
#> 2 b10    2      2011
#> 3 e15    1      2015
```

如果某个特定部分没有用，你可以使用 `NA` 作为列名来在结果中省略它：

```
df3 |>
  separate_wider_delim(
    x,
    delim = ".",
    names = c("code", NA, "year")
  )
#> # A tibble: 3 x 2
#>   code  year
#>   <chr> <chr>
```

## 14.4 从字符串提取数据

```
#> 1 a10    2022  
#> 2 b10    2011  
#> 3 e15    2015
```

`separate_wider_position()` 的工作方式略有不同, 因为你通常需要指定每个列的宽度。因此, 你给它一个命名的整数向量, 其中名称给出新列的名称, 而值是它所占据的字符数。你可以通过不命名某些值来从输出中省略它们:

```
df4 <- tibble(x = c("202215TX", "202122LA", "202325CA"))  
df4 |>  
  separate_wider_position(  
    x,  
    widths = c(year = 4, age = 2, state = 2)  
  )  
#> # A tibble: 3 x 3  
#>   year   age   state  
#>   <chr> <chr> <chr>  
#> 1 2022   15    TX  
#> 2 2021   22    LA  
#> 3 2023   25    CA
```

### 14.4.3 诊断扩展问题

`separate_wider_delim()`<sup>6</sup> 需要一个固定且已知数量的列。如果某些行没有预期数量的组件怎么办? 可能存在两种问题, 即组件太少或太多, 因此

---

<sup>6</sup>同样的原则也适用于 `separate_wider_position()` 和 `separate_wider_regex()`。

## 14 字符串

`separate_wider_delim()` 提供了 `too_few` 和 `too_many` 两个参数来帮助处理这些问题。首先，让我们使用以下示例数据集来查看 `too_few` 的情况：

```
df <- tibble(x = c("1-1-1", "1-1-2", "1-3", "1-3-2", "1"))

df |>
  separate_wider_delim(
    x,
    delim = "-",
    names = c("x", "y", "z")
  )

#> Error in `separate_wider_delim()`:
#> ! Expected 3 pieces in each element of `x`.
#> ! 2 values were too short.
#> i Use `too_few = "debug"` to diagnose the problem.
#> i Use `too_few = "align_start"/"align_end"` to silence this message.
```

你会得到一个错误提示，但错误消息给出了一些关于如何继续进行的建议。  
让我们开始调试这个问题：

```
debug <- df |>
  separate_wider_delim(
    x,
    delim = "-",
    names = c("x", "y", "z"),
    too_few = "debug"
  )

#> Warning: Debug mode activated: adding variables `x_ok`, `x_pieces`, and
```

## 14.4 从字符串提取数据

```
#> `x_remainder`.  
debug  
#> # A tibble: 5 x 6  
#>   x     y     z   x_ok  x_pieces x_remainder  
#>   <chr> <chr> <chr> <lgl>    <int> <chr>  
#> 1 1-1-1 1     1     TRUE      3 ""  
#> 2 1-1-2 1     2     TRUE      3 ""  
#> 3 1-3     3     <NA>    FALSE      2 ""  
#> 4 1-3-2 3     2     TRUE      3 ""  
#> 5 1     <NA>  <NA>  FALSE      1 ""
```

当使用调试模式时，你会在输出中看到添加了三个额外的列：`x_ok`、`x_pieces` 和 `x_remainder`（如果你使用不同名称的变量进行拆分，你会得到一个不同的前缀）。在这里，`x_ok` 允许你快速找到那些失败的输入：

```
debug |> filter(!x_ok)  
#> # A tibble: 2 x 6  
#>   x     y     z   x_ok  x_pieces x_remainder  
#>   <chr> <chr> <chr> <lgl>    <int> <chr>  
#> 1 1-3     3     <NA>  FALSE      2 ""  
#> 2 1     <NA>  <NA>  FALSE      1 ""
```

`x_pieces` 告诉我们找到了多少组件。与预期的 3 个 (`names` 的长度) 相比，当得到的组件数量少于预期时，`x_remainder` 并不太有用，但稍后我们会再次看到它。

有时查看这些调试信息会发现你的分隔符策略有问题，或者建议你在拆分之前需要进行更多的预处理。在这种情况下，解决上游问题并确保移除 `too_few = "debug"` 以确保新问题变成错误。

## 14 字符串

在其他情况下，你可能想用 NAs 填充缺失的组件并继续。这就是 `too_few = "align_start"` 和 `too_few = "align_end"` 的作用，它们允许你控制 NAs 应该放在哪里：

```
df |>
  separate_wider_delim(
    x,
    delim = "-",
    names = c("x", "y", "z"),
    too_few = "align_start"
)
#> # A tibble: 5 x 3
#>   x     y     z
#>   <chr> <chr> <chr>
#> 1 1     1     1
#> 2 1     1     2
#> 3 1     3     <NA>
#> 4 1     3     2
#> 5 1     <NA>  <NA>
```

如果你有太多的组件，同样的原则也适用：

```
df <- tibble(x = c("1-1-1", "1-1-2", "1-3-5-6", "1-3-2", "1-3-5-7-9"))

df |>
  separate_wider_delim(
    x,
    delim = "-",
```

## 14.4 从字符串提取数据

```
names = c("x", "y", "z")
)
#> Error in `separate_wider_delim()`:
#> ! Expected 3 pieces in each element of `x`.
#> ! 2 values were too long.
#> i Use `too_many = "debug"` to diagnose the problem.
#> i Use `too_many = "drop"/"merge"` to silence this message.
```

但是现在，当我们调试结果时，你可以看到 `x_remainder` 的目的：

```
debug <- df |>
  separate_wider_delim(
    x,
    delim = "-",
    names = c("x", "y", "z"),
    too_many = "debug"
  )
#> Warning: Debug mode activated: adding variables `x_ok`, `x_pieces`, and
#> `x_remainder`.
debug |> filter(!x_ok)
#> # A tibble: 2 x 6
#>   x         y     z     x_ok x_pieces x_remainder
#>   <chr>     <chr> <chr> <lgl>    <int> <chr>
#> 1 1-3-5-6   3     5     FALSE      4 -6
#> 2 1-3-5-7-9 3     5     FALSE      5 -7-9
```

处理过多组件时，会有一些稍微不同的选项：你可以选择静默地“删除”任何额外的组件，或者将它们全部“合并”到最后一列：

## 14 字符串

```
df |>
  separate_wider_delim(
    x,
    delim = "-",
    names = c("x", "y", "z"),
    too_many = "drop"
  )
#> # A tibble: 5 x 3
#>   x     y     z
#>   <chr> <chr> <chr>
#> 1 1     1     1
#> 2 1     1     2
#> 3 1     3     5
#> 4 1     3     2
#> 5 1     3     5
```

```
df |>
  separate_wider_delim(
    x,
    delim = "-",
    names = c("x", "y", "z"),
    too_many = "merge"
  )
#> # A tibble: 5 x 3
#>   x     y     z
#>   <chr> <chr> <chr>
```

```
#> 1 1      1      1
#> 2 1      1      2
#> 3 1      3      5-6
#> 4 1      3      2
#> 5 1      3      5-7-9
```

## 14.5 字母

在本节中，我们将介绍一些函数，这些函数允许你处理字符串中的单个字母。你将学习如何查找字符串的长度、提取子字符串以及在图和表中处理长字符串。

### 14.5.1 长度

`str_length()` 告诉你字符串中字母的数量：

```
str_length(c("a", "R for data science", NA))
#> [1] 1 18 NA
```

你可以使用 `count()` 来查找美国婴儿名字长度的分布情况，然后使用 `filter()` 来查看最长的名字，这些名字恰好有 15 个字母<sup>7</sup>。

```
babynames |>
  count(length = str_length(name), wt = n)
#> # A tibble: 14 x 2
#>   length     n
```

---

<sup>7</sup>看看这些条目，我们猜测 `babynames` 数据会省略空格或连字符，并在 15 个字母之后截断。

## 14 字符串

```
#>      <int>    <int>
#> 1      2  338150
#> 2      3  8589596
#> 3      4  48506739
#> 4      5  87011607
#> 5      6  90749404
#> 6      7  72120767
#> # i 8 more rows

babynames |>
  filter(str_length(name) == 15) |>
  count(name, wt = n, sort = TRUE)
#> # A tibble: 34 x 2
#>   name              n
#>   <chr>            <int>
#> 1 Franciscojavier 123
#> 2 Christopherjohn 118
#> 3 Johnchristopher 118
#> 4 Christopherjame 108
#> 5 Christophermich 52
#> 6 Ryanchristopher 45
#> # i 28 more rows
```

### 14.5.2 提取子集

你可以使用 `str_sub(string, start, end)` 来提取字符串的一部分，其中 `start` 和 `end` 是子字符串开始和结束的位置。`start` 和 `end` 参数是包含性

的，因此返回的字符串的长度将是 `end - start + 1`。

```
x <- c("Apple", "Banana", "Pear")
str_sub(x, 1, 3)
#> [1] "App" "Ban" "Pea"
```

你也可以使用负值来从字符串的末尾开始计数：-1 是最后一个字符，-2 是倒数第二个字符，依此类推。

```
str_sub(x, -3, -1)
#> [1] "ple" "ana" "ear"
```

请注意，如果字符串太短，`str_sub()` 不会失败，它将返回尽可能多的字符：

```
str_sub("a", 1, 5)
#> [1] "a"
```

我们可以使用 `str_sub()` 与 `mutate()` 来找到每个名字的首字母和末字母：

```
babynames |>
  mutate(
    first = str_sub(name, 1, 1),
    last = str_sub(name, -1, -1)
  )
#> # A tibble: 1,924,665 x 7
#>   year sex     name      n    prop first last
#>   <dbl> <fct>   <chr> <dbl> <dbl> <chr> <chr>
```

## 14 字符串

```
#>   <dbl> <chr> <chr>      <int> <dbl> <chr> <chr>
#> 1 1880 F     Mary        7065 0.0724 M     y
#> 2 1880 F     Anna        2604 0.0267 A     a
#> 3 1880 F     Emma        2003 0.0205 E     a
#> 4 1880 F     Elizabeth  1939 0.0199 E     h
#> 5 1880 F     Minnie      1746 0.0179 M     e
#> 6 1880 F     Margaret    1578 0.0162 M     t
#> # i 1,924,659 more rows
```

### 14.5.3 练习

1. 在计算婴儿名字长度的分布时，为什么我们使用 `wt = n`？
2. 使用 `str_length()` 和 `str_sub()` 从每个婴儿的名字中提取中间的字母。如果字符串有偶数个字符，你会怎么做？
3. 随着时间的推移，`babynames` 的长度有什么主要趋势吗？首字母和尾字母的流行程度如何？

## 14.6 非英语文本

到目前为止，我们主要关注英文文本，这是因为两个原因使其特别易于处理。首先，英文字母表相对简单：只有 26 个字母。其次（也许是更重要的），我们今天使用的计算基础设施主要是由英语使用者设计的。不幸的是，我们没有篇幅全面介绍非英语语言。不过，我们还是想提醒你们注意可能会遇到的一些最大挑战：编码、字母变化和依赖于地区的函数。

### 14.6.1 编码

在处理非英文文本时，第一个挑战通常是编码。为了了解正在发生的事情，我们需要深入了解计算机如何表示字符串。在 R 中，我们可以使用 `charToRaw()` 来获取字符串的底层表示：

```
charToRaw("Hadley")
#> [1] 48 61 64 6c 65 79
```

这六个十六进制数字中的每一个都代表一个字母：48 是 H，61 是 a，以此类推。从十六进制数字到字符的映射称为编码，在这种情况下，这种编码称为 ASCII。ASCII 在表示英文字符方面做得很好，因为它是美国信息交换标准代码（American Standard Code for Information Interchange）。

对于非英文语言来说，事情就没那么简单了。在计算机发展的早期，有许多相互竞争的标准用于编码非英文字符。例如，欧洲有两种不同的编码：Latin1（也称为 ISO-8859-1）用于西欧语言，而 Latin2（也称为 ISO-8859-2）用于中欧语言。在 Latin1 中，字节 b1 是 “±”，但在 Latin2 中，它是 “¤”！幸运的是，今天几乎在所有地方都支持一个标准：UTF-8。UTF-8 可以编码今天人类使用的几乎所有字符，以及许多额外的符号，如表情符号。

`readr` 在所有地方都使用 UTF-8。这是一个很好的默认设置，但对于不使用 UTF-8 的旧系统生成的数据将会失败。如果发生这种情况，当你输出字符串时，它们看起来会很奇怪。有时只是一个或两个字符可能会出错；在其他时候，你会得到完全乱码。例如，以下是两个具有异常编码的内联 CSV<sup>8</sup>：

---

<sup>8</sup>这里我使用特殊的\x 将二进制数据直接编码为字符串。

## 14 字符串

```
x1 <- "text\nEl Ni\xf1o was particularly bad this year"
read_csv(x1)$text
#> [1] "El Ni\xf1o was particularly bad this year"

x2 <- "text\n\x82\xb1\x82\xf1\x82\xc9\x82\xbf\x82\xcd"
read_csv(x2)$text
#> [1] "\x82\xb1\x82\xf1\x82\xbf\x82\xcd"
```

要正确读取这些，您可以通过 `locale` 参数指定编码：

```
read_csv(x1, locale = locale(encoding = "Latin1"))$text
#> [1] "El Ni\u00f1o was particularly bad this year"

read_csv(x2, locale = locale(encoding = "Shift-JIS"))$text
#> [1] " こんにちは"
```

如何找到正确的编码？如果你很幸运，它可能会在数据文档中的某个地方被提及。不幸的是，这种情况很少见，所以 `readr` 提供了 `guess_encoding()` 来帮助你找出它。这并不是万无一失的，并且在有大量文本时（与这里不同）效果会更好，但这是一个合理的起点。预计在你找到正确的编码之前，需要尝试几种不同的编码。

编码是一个丰富且复杂的话题；我们在这里只是触及了皮毛。如果你想了解更多，我们推荐阅读 <http://kunststube.net/encoding/> 上的详细解释。

### 14.6.2 字母变体

在使用带有重音的语言时，确定字母的位置（例如使用 `str_length()` 和 `str_sub()`）会面临重大挑战，因为重音字母可能会被编码为一个单独的字符（例如，ü）或者通过将不带重音的字母（例如，u）与变音符号（例如，”）组合成两个字符来表示。例如，以下代码展示了两种看起来相同的表示 ü 的方式：

```
u <- c("\u00fc", "u\u0308")
str_view(u)
#> [1] | ü
#> [2] | ü
```

但是两个字符串的长度不同，它们的第一个字符也不同：

```
str_length(u)
#> [1] 1 2
str_sub(u, 1, 1)
#> [1] "ü" "u"
```

最后，请注意，使用 `==` 来比较这些字符串时，它们会被解释为不同的字符串，而 `stringr` 包中的 `str_equal()` 函数则会识别出两者具有相同的外观。

```
u[[1]] == u[[2]]
#> [1] FALSE

str_equal(u[[1]], u[[2]])
#> [1] TRUE
```

### 14.6.3 区域依赖函数

最后，有一些 `stringr` 函数的行为取决于你的区域（locale）。区域类似于一种语言，但包括一个可选的区域标识符来处理语言内的地区差异。区域由小写语言缩写指定，后面可以选择性地跟上一个下划线和一个大写区域标识符。例如，“en”代表英语，“en\_GB”代表英国英语，而“en\_US”代表美国英语。如果你还不知道你的语言的代码，[维基百科](#)有一个很好的列表，你可以通过查看 `stringi::stri_locale_list()` 来查看 `stringr` 支持哪些区域。

基础 R 的字符串函数会自动使用你的操作系统设置的区域。这意味着基础 R 的字符串函数会按照你期望的方式处理你的语言，但如果你与来自不同国家的人分享你的代码，代码的行为可能会有所不同。为了避免这个问题，`stringr` 默认使用“en”区域（即英语规则），并要求你指定 `locale` 参数来覆盖它。幸运的是，只有两组函数在处理时需要考虑区域：改变大小写和排序。

改变大小写的规则在不同语言之间有所不同。例如，土耳其语有两种 i：带点和不带点。由于它们是两个不同的字母，所以它们的大写形式也不同：

```
str_to_upper(c("i", "ı"))
#> [1] "I" "I"
str_to_upper(c("i", "ı"), locale = "tr")
#> [1] "İ" "I"
```

字符串排序依赖于字母表的顺序，而字母表的顺序并不是每种语言都相同的<sup>9</sup>！举个例子：在捷克语中，“ch”是一个复合字母，它在字母表中出现在字母 h 之后。

---

<sup>9</sup>在没有字母表的语言中排序，比如中文，就更加复杂了。

```
str_sort(c("a", "c", "ch", "h", "z"))
#> [1] "a"   "c"   "ch"  "h"   "z"
str_sort(c("a", "c", "ch", "h", "z"), locale = "cs")
#> [1] "a"   "c"   "h"   "ch"  "z"
```

This also comes up when sorting strings with `dplyr::arrange()`, which is why it also has a `locale` argument.

## 14.7 小结

在本章中，你已经了解了 `stringr` 包的一些强大功能：如何创建、组合和提取字符串，以及在使用非英文字符串时可能遇到的一些挑战。现在是时候学习处理字符串的最重要和强大的工具之一：正则表达式了。正则表达式是一种非常简洁但非常富有表现力的语言，用于描述字符串中的模式，其是下一章讨论的主题。



# 15 正则表达式

## 15.1 引言

在章节 ?? 部分，你学习了一系列用于处理字符串的有用函数。本章将重点介绍使用正则表达式的函数，正则表达式（regular expression）是一种简洁而强大的语言，用于描述字符串中的模式。术语“正则表达式”有点长，所以大多数人将其缩写为“regex”<sup>1</sup>或“regexp”。

本章首先介绍正则表达式的基础知识以及数据分析中最有用的 `stringr` 函数。然后，我们将扩展你对模式的了解，并介绍七个重要的新主题（转义、锚定、字符类、简写类、量词、优先级和分组）。接下来，我们将讨论 `stringr` 函数可以处理的其他类型的模式，以及允许你调整正则表达式操作的各种“标志”。最后，我们将概述 tidyverse 和 base R 中可能会使用正则表达式的其他地方。

### 15.1.1 必要条件

在本章中，我们将使用 tidyverse 的核心成员 `stringr` 和 `tidyverse` 中的正则表达式函数，以及 `babynames` 包中的数据。

---

<sup>1</sup>你可以用硬音 g (reg-x) 或软音 g (rej-x) 来发音。

```
library(tidyverse)
library(babynames)
```

在本章中，我们将使用非常简单的内联示例的混合方式，以便你能够理解基本概念，还会使用 `babynames` 数据集中的数据，以及来自 `stringr` 的三个字符向量：

- `fruit` 包含了 80 种水果的名称。
- `words` 包含了 980 个常见的英语单词。
- `sentences` 包含了 720 个短句。

## 15.2 模式的基础

我们将使用 `str_view()` 来学习正则表达式模式是如何工作的。在上一章中，我们使用 `str_view()` 来更好地理解字符串与其输出表示之间的区别，现在我们将使用它的第二个参数，即正则表达式。当提供这个参数时，`str_view()` 将仅显示字符串向量中匹配的部分，将每个匹配项用 `< >` 括起来，并在可能的情况下将匹配项以蓝色高亮显示。

最简单的模式由字母和数字组成，它们会精确匹配这些字符：

```
str_view(fruit, "berry")
#> [6] | bil<berry>
#> [7] | black<berry>
#> [10] | blue<berry>
#> [11] | boysen<berry>
#> [19] | cloud<berry>
```

```
#> [21] | cran<berry>
#> ... and 8 more
```

字母和数字进行精确匹配，被称为字面字符 (literal characters)。大多数标点符号字符，如 .、+、\*、[、] 和 ? 具有特殊含义<sup>2</sup>，被称为元字符 (metacharacters)。例如，. 将匹配任何字符<sup>3</sup>，所以 "a." 将匹配任何包含 “a” 后面跟着另一个字符的字符串：

```
str_view(c("a", "ab", "ae", "bd", "ea", "eab"), "a.")
#> [2] | <ab>
#> [3] | <ae>
#> [6] | e<ab>
```

或者，我们可以找到所有包含 “a”，后跟三个字母，再后跟 “e” 的水果：

```
str_view(fruit, "a...e")
#> [1] | <apple>
#> [7] | bl<ackbe>rry
#> [48] | mand<arine>
#> [51] | nect<arine>
#> [62] | pine<apple>
#> [64] | pomegr<anate>
#> ... and 2 more
```

**量词 (Quantifiers)** 控制模式可以匹配的次数：

---

<sup>2</sup>你将在小节 ?? 中学习如何转义这些特殊含义。

<sup>3</sup>除了\n 以外的任何字符。

## 15 正则表达式

- `?` 使得一个模式变为可选的（即它匹配 0 次或 1 次）
- `+` 允许一个模式重复（即它至少匹配一次）
- `*` 允许一个模式变为可选的或重复（即它匹配任意次数，包括 0 次）

```
# ab? matches an "a", optionally followed by a "b".
str_view(c("a", "ab", "abb"), "ab?")
#> [1] | <a>
#> [2] | <ab>
#> [3] | <ab>b

# ab+ matches an "a", followed by at least one "b".
str_view(c("a", "ab", "abb"), "ab+")
#> [2] | <ab>
#> [3] | <abb>

# ab* matches an "a", followed by any number of "b"s.
str_view(c("a", "ab", "abb"), "ab*")
#> [1] | <a>
#> [2] | <ab>
#> [3] | <abb>
```

字符类 (Character classes) 由 `[]` 定义，允许你匹配一组字符，例如：`[abcd]` 匹配“a”、“b”、“c”或“d”。你也可以通过在开头使用 `^` 来反转匹配：`[^abcd]` 匹配除“a”、“b”、“c”或“d”之外的任何字符。我们可以利用这个思路来查找被元音字母包围的“x”，或者被辅音字母包围的“y”。

```
str_view(words, "[aeiou]x[aeiou]")
#> [284] | <exa>ct
```

## 15.2 模式的基础

```
#> [285] | <exa>mple
#> [288] | <exe>rcise
#> [289] | <exi>st
str_view(words, "[^aeiou]y[^aeiou]")
#> [836] | <sys>tem
#> [901] | <typ>e
```

你可以使用分隔符（**alternation**）| 在一个或多个备选模式中进行选择。例如，下面的模式会查找包含“apple”、“melon”或“nut”，或者一个重复元音字母的水果。

```
str_view(fruit, "apple|melon|nut")
#> [1] | <apple>
#> [13] | canary <melon>
#> [20] | coco<nut>
#> [52] | <nut>
#> [62] | pine<apple>
#> [72] | rock <melon>
#> ... and 1 more
str_view(fruit, "aa|ee|ii|oo|uu")
#> [9] | bl<oo>d orange
#> [33] | g<oo>seberry
#> [47] | lych<ee>
#> [66] | purple mangost<ee>n
```

正则表达式非常紧凑，使用了大量标点符号字符，所以一开始可能会显得让人难以理解和难以阅读。不过不用担心，随着不断实践，你会越来越熟练，简

## 15 正则表达式

单的模式很快就会变得驾轻就熟。让我们通过练习一些有用的 `stringr` 函数来开始这个过程吧。

### 15.3 关键函数

既然你已经掌握了正则表达式的基础知识，接下来就让我们使用 `stringr` 和 `tidyverse` 函数来应用它们吧。在以下部分，你将学习如何检测匹配项是否存在，如何计算匹配项的数量，如何用固定文本替换匹配项，以及如何使用模式来提取文本。

#### 15.3.1 检测匹配项

`str_detect()` 函数返回一个逻辑向量，如果模式与字符向量中的某个元素匹配，则返回 `TRUE`，否则返回 `FALSE`:

```
str_detect(c("a", "b", "c"), "[aeiou]")
#> [1] TRUE FALSE FALSE
```

由于 `str_detect()` 返回一个与初始向量长度相同的逻辑向量，因此能与 `filter()` 搭配很好。例如，这段代码用于查找所有包含小写字母“x”的最受欢迎的名字:

```
babynames |>
  filter(str_detect(name, "x")) |>
  count(name, wt = n, sort = TRUE)
#> # A tibble: 974 x 2
```

### 15.3 关键函数

```
#>   name          n
#>   <chr>     <int>
#> 1 Alexander  665492
#> 2 Alexis      399551
#> 3 Alex        278705
#> 4 Alexandra  232223
#> 5 Max         148787
#> 6 Alexa       123032
#> # i 968 more rows
```

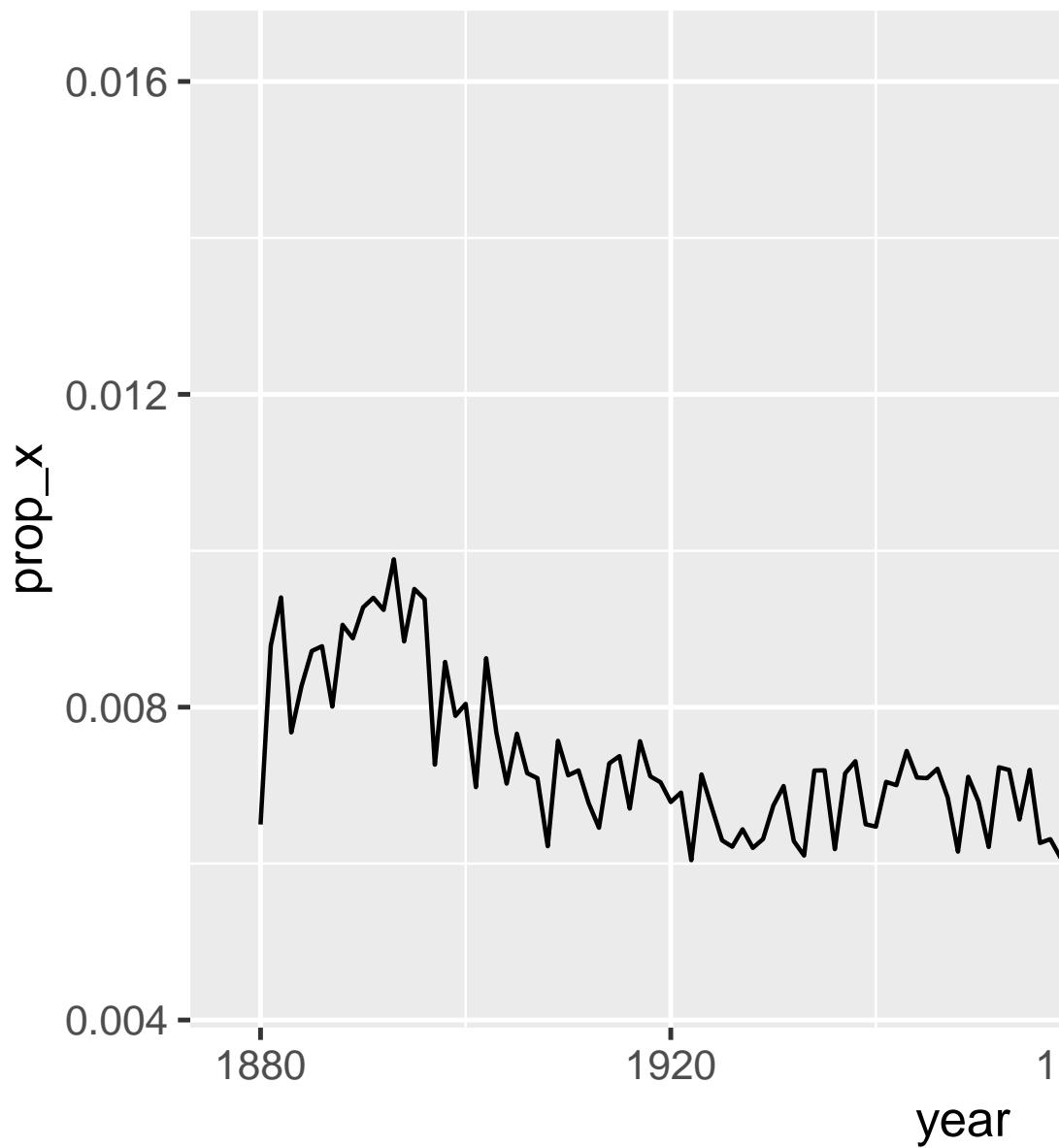
还可以通过将 `str_detect()` 与 `summarize()` 结合，并搭配 `sum()` 或 `mean()` 来使用。`sum(str_detect(x, pattern))` 告诉你匹配的观测的数量，而 `mean(str_detect(x, pattern))` 告诉你匹配的比例。例如，以下代码段计算和可视化了按年份划分的婴儿名字<sup>4</sup> 包含”x”的比例。看来最近它们的受欢迎程度大幅增加了！

```
babynames |>
  group_by(year) |>
  summarize(prop_x = mean(str_detect(name, "x"))) |>
  ggplot(aes(x = year, y = prop_x)) +
  geom_line()
```

---

<sup>4</sup>这给出了包含”x”的名字的比例；如果你想要知道名字中包含”x”的婴儿的比例，你需要计算一个加权平均值。

15 正则表达式



与 `str_detect()` 密切相关的两个函数是 `str_subset()` 和 `str_which()`。  
`str_subset()` 返回一个只包含匹配字符串的字符向量；`str_which()` 返回一个给出匹配字符串位置的整数向量。

### 15.3.2 匹配次数

从复杂度来看，比 `str_detect()` 更进一步的是 `str_count()`：它不仅仅告诉你匹配与否，还会告诉你每个字符串中有多少匹配项。

```
x <- c("apple", "banana", "pear")
str_count(x, "p")
#> [1] 2 0 1
```

请注意，每个匹配都从上一个匹配的末尾开始，即正则表达式匹配永远不会重叠。例如，在"abababa" 中，"aba" 模式将匹配多少次？正则表达式说是两次，而不是三次：

```
str_count("abababa", "aba")
#> [1] 2
str_view("abababa", "aba")
#> [1] | <aba>b<aba>
```

`str_count()` 与 `mutate()` 结合使用是很自然的。下面的示例使用 `str_count()` 与字符类来计算每个名字中的元音和辅音字母的数量。

## 15 正则表达式

```
babynames |>
  count(name) |>
  mutate(
    vowels = str_count(name, "[aeiou]"),
    consonants = str_count(name, "[^aeiou]")
  )
#> # A tibble: 97,310 x 4
#>   name      n vowels consonants
#>   <chr>    <int>  <int>     <int>
#> 1 Aaban     10     2         3
#> 2 Aabha      5     2         3
#> 3 Aabid      2     2         3
#> 4 Aabir      1     2         3
#> 5 Abriella    5     4         5
#> 6 Aada       1     2         2
#> # i 97,304 more rows
```

如果仔细观察，你会注意到我们的计算有些问题：“Aaban”包含三个“a”，但我们的汇报只显示了两个元音字母。这是因为正则表达式是区分大小写的。有三种方法可以修复这个问题：

- 将大写元音字母添加到字符类中：

```
str_count(name, "[aeiouAEIOU]").
```

- 告诉正则表达式忽略大小写：

```
str_count(name, regex("[aeiou]", ignore_case = TRUE)).
```

我们将在小节 ?? 讨论更多内容。

### 15.3 关键函数

- 使用 `str_to_lower()` 将名字转换为小写：

```
str_count(str_to_lower(name), "[aeiou"])。
```

在处理字符串时，这种多样化的方法是非常典型的——通常有多种方式可以达到你的目标，要么是通过使你的模式更复杂，要么是对你的字符串进行一些预处理。如果你在使用一种方法时遇到困难，从另一个角度解决问题往往是有用的。

在这种情况下，由于我们对名字应用了两个函数，我认为先转换它更容易：

```
babynames |>
  count(name) |>
  mutate(
    name = str_to_lower(name),
    vowels = str_count(name, "[aeiou"]),
    consonants = str_count(name, "[^aeiou"])
  )
#> # A tibble: 97,310 x 4
#>   name      n  vowels  consonants
#>   <chr>  <int>  <int>     <int>
#> 1 aaban     10      3        2
#> 2 aabha      5      3        2
#> 3 aabid      2      3        2
#> 4 aabir      1      3        2
#> 5 aabriella   5      5        4
#> 6 aada       1      3        1
#> # i 97,304 more rows
```

### 15.3.3 替换值

除了检测和计算匹配项之外，我们还可以使用 `str_replace()` 和 `str_replace_all()` 来修改它们。`str_replace()` 替换第一个匹配项，如其名所示；`str_replace_all()` 则会替换所有匹配项。

```
x <- c("apple", "pear", "banana")
str_replace_all(x, "[aeiou]", "-")
#> [1] "-ppl-"  "p--r"   "b-n-n-"
```

`str_remove()` 和 `str_remove_all()` 是方便的快捷方式，用于 `str_replace(x, pattern, "")`：

```
x <- c("apple", "pear", "banana")
str_remove_all(x, "[aeiou]")
#> [1] "ppl" "pr"   "bnn"
```

在进行数据清洗时，这些函数通常会与 `mutate()` 一起使用，并且你会经常反复应用它们来消除不一致格式的多层结构。

### 15.3.4 提取变量

我们要讨论的最后一个函数 `separate_wider_regex()` 使用正则表达式将数据从一个列提取到一个或多个新列中。它与你在小节 ?? 中了解的 `separate_wider_position()` 和 `separate_wider_delim()` 是同类函数。这些函数存在于 `tidyverse` 中，因为它们操作的是数据框（的列），而不是单独的向量。

让我们创建一个简单的数据集来展示它的工作原理。这里我们有一些从 `babynames` 派生的数据，其中包含了一些人的名字、性别和年龄，但这些数据的格式相当奇怪的<sup>5</sup>：

```
df <- tribble(
  ~str,
  "<Sheryl>-F_34",
  "<Kisha>-F_45",
  "<Brandon>-N_33",
  "<Sharon>-F_38",
  "<Penny>-F_58",
  "<Justin>-M_41",
  "<Patricia>-F_84",
)
```

要使用 `separate_wider_regex()` 提取这些数据，我们只需要构建一系列与每个部分匹配的正则表达式。如果希望该部分内容出现在输出中，给它指定一个名称：

```
df |>
  separate_wider_regex(
    str,
    patterns = c(
      "<",
      name = "[A-Za-z]+",
      ">-",
    ))
```

---

<sup>5</sup> 我们希望能向你保证，在现实生活中你永远不会看到这种奇怪的数据格式；但不幸的是，在你的职业生涯中，你可能会看到比这更奇怪的数据格式！

## 15 正则表达式

```
gender = ".",
"_" ,
age = "[0-9]"+
)
)

#> # A tibble: 7 x 3
#>   name    gender age
#>   <chr>   <chr> <chr>
#> 1 Sheryl  F     34
#> 2 Kisha   F     45
#> 3 Brandon N     33
#> 4 Sharon  F     38
#> 5 Penny   F     58
#> 6 Justin  M     41
#> # i 1 more row
```

如果匹配失败，你可以使用 `too_few = "debug"` 来找出问题所在，就像 `separate_wider_delim()` 和 `separate_wider_position()` 一样。

### 15.3.5 练习

1. 哪个婴儿名字中含有的元音字母最多？哪个名字的元音字母比例最高？（提示：分母是什么？）
2. 将 "a/b/c/d/e" 中的所有正斜杠 (/) 替换为反斜杠 (\)。如果试图通过将所有反斜杠替换为正斜杠来撤销这个转换会发生什么？（我们很快就会讨论这个问题。）

3. 使用 `str_replace_all()` 实现一个简单的 `str_to_lower()` 版本。
4. 创建一个正则表达式，用来匹配你所在国家常见的电话号码书写方式。

## 15.4 模式的细节

现在你已经了解了模式语言的基础知识，以及如何将其与一些 `stringr` 和 `tidyR` 函数一起使用，现在是时候深入了解更多细节了。首先，我们将从转义 (**escaping**) 开始，它允许你匹配通常会被特殊处理的元字符。接下来，你将学习锚点 (**anchors**)，它允许你匹配字符串的开始或结束。然后，你将更深入地了解字符类 (**character classes**) 和它们的快捷方式，它们允许你匹配集合中的任何字符。接着，你将学习量词 (**quantifiers**) 的最后一些细节，它们控制模式可以匹配多少次。之后，我们必须覆盖重要 (但复杂) 的主题，即运算符优先级 (**operator precedence**) 和括号。最后，我们将以模式分组 (**grouping**) 组件的一些细节结束。

我们在这里使用的术语是每个组件的技术名称。它们并不总是最能体现其目的，但如果你以后想在网上搜索更多细节，知道正确的术语会很有帮助。

### 15.4.1 转义

为了匹配字面上的`.`，你需要一个转义符，它告诉正则表达式要从字面上匹配元字符<sup>6</sup>。和字符串一样，正则表达式使用反斜杠进行转义。因此，要匹配`.`，正则表达式为`\.`。不幸的是这会造成一个问题。我们使用字符串来表示正则表达式，而\也在字符串中用作转义符。因此，要创建正则表达式`\.`，我们需要字符串"`\.\.`"，如下例所示。

---

<sup>6</sup>元字符的完备集为 `.^$\\|*+?{}[]()`

## 15 正则表达式

```
# To create the regular expression \., we need to use \\.
dot <- "\\."

# But the expression itself only contains one \
str_view(dot)
#> [1] | \.

# And this tells R to look for an explicit .
str_view(c("abc", "a.c", "bef"), "a\\\\.c")
#> [2] | <a.c>
```

在这本书中，我们通常会在没有引号的情况下编写正则表达式，比如\..。如果我们需要强调实际输入的内容，我们会用引号将其括起来并添加额外的转义字符，比如"\\".。

如果\在正则表达式中被用作转义字符，那么如何匹配一个实际的\呢？你需要对它进行转义，创建正则表达式\\。为了创建这个正则表达式，你需要使用字符串，而字符串本身也需要对\进行转义。这意味着要匹配一个实际的\，你需要写"\\\\\\\"——你需要四个反斜杠来匹配一个！

```
x <- "a\\b"
str_view(x)
#> [1] | a\b
str_view(x, "\\\\\\\")
#> [1] | a<\>b
```

或者，你可能发现使用小节 ?? 中学到的原始字符串会更加简单。这样可以让你避免一层的转义：

```
str_view(x, r"\{\}\}")
#> [1] | a<\>b
```

如果你试图匹配一个字面上的.、\$、|、\*、+、?、{、}、(或)，除了使用反斜杠转义之外还有一个替代方案：你可以使用字符类：.、\$、|、... 都匹配其字面值。

```
str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c")
#> [2] | <a.c>
str_view(c("abc", "a.c", "a*c", "a c"), ".[*]c")
#> [3] | <a*c>
```

### 15.4.2 锚点

默认情况下，正则表达式会匹配字符串的任何部分。如果你想在开头或结尾处进行匹配，你需要使用 ^ 来锚定正则表达式的开始，或者使用 \$ 来锚定正则表达式的结束：

```
str_view(fruit, "^a")
#> [1] | <a>pple
#> [2] | <a>pricot
#> [3] | <a>ocado
str_view(fruit, "a$")
#> [4] | banan<a>
#> [15] | cherimoy<a>
#> [30] | feijo<a>
#> [36] | guav<a>
```

## 15 正则表达式

```
#> [56] | papay<a>
#> [74] | satsum<a>
```

可能会让人误以为 `$` 应该匹配字符串的开始，因为我们常常这样写金额，但这并不是正则表达式所期望的。

要强制正则表达式仅匹配整个字符串，需要用 `^` 和 `$` 同时进行锚定：

```
str_view(fruit, "apple")
#> [1] | <apple>
#> [62] | pine<apple>
str_view(fruit, "^apple$")
#> [1] | <apple>
```

你也可以使用`\b` 来匹配单词之间的边界（即单词的开始或结束）。这在使用 RStudio 的查找和替换工具时特别有用。例如，如果你想查找所有 `sum()` 的用法，你可以搜索`\bsum\b` 来避免匹配到 `summarize`、`summary`、`rowsum` 等单词：

```
x <- c("summary(x)", "summarize(df)", "rowsum(x)", "sum(x)")
str_view(x, "sum")
#> [1] | <sum>mary(x)
#> [2] | <sum>marize(df)
#> [3] | row<sum>(x)
#> [4] | <sum>(x)
str_view(x, "\bsum\b")
#> [4] | <sum>(x)
```

## 15.4 模式的细节

当单独使用时，锚点会产生一个零宽度的匹配：

```
str_view("abc", c("$", "^", "\b"))
#> [1] | abc<>
#> [2] | <>abc
#> [3] | <>abc<>
```

这有助于你理解当替换一个单独的锚点时会发生什么：

```
str_replace_all("abc", c("$", "^", "\b"), "--")
#> [1] "abc--"    "--abc"    "--abc--"
```

### 15.4.3 字符类

字符类（或字符集）允许你匹配集合中的任何字符。如上所述，你可以使用 `[]` 来构建自己的集合，其中 `[abc]` 匹配“a”、“b”或“c”，而 `[^abc]` 匹配除了“a”、“b”或“c”之外的任何字符。除了 `^` 之外，`[]` 内还有两个字符具有特殊含义：

- `-` 定义了一个范围，例如，`[a-z]` 匹配任何小写字母，而 `[0-9]` 匹配任何数字。
- `\` 用于转义特殊字符，因此 `[\^-\`]` 匹配字符 `^`、`-`或```。

这里有几个例子

```
x <- "abcd ABCD 12345 -!@#%."
str_view(x, "[abc]+")
#> [1] | <abc>d ABCD 12345 -!@#%.
```