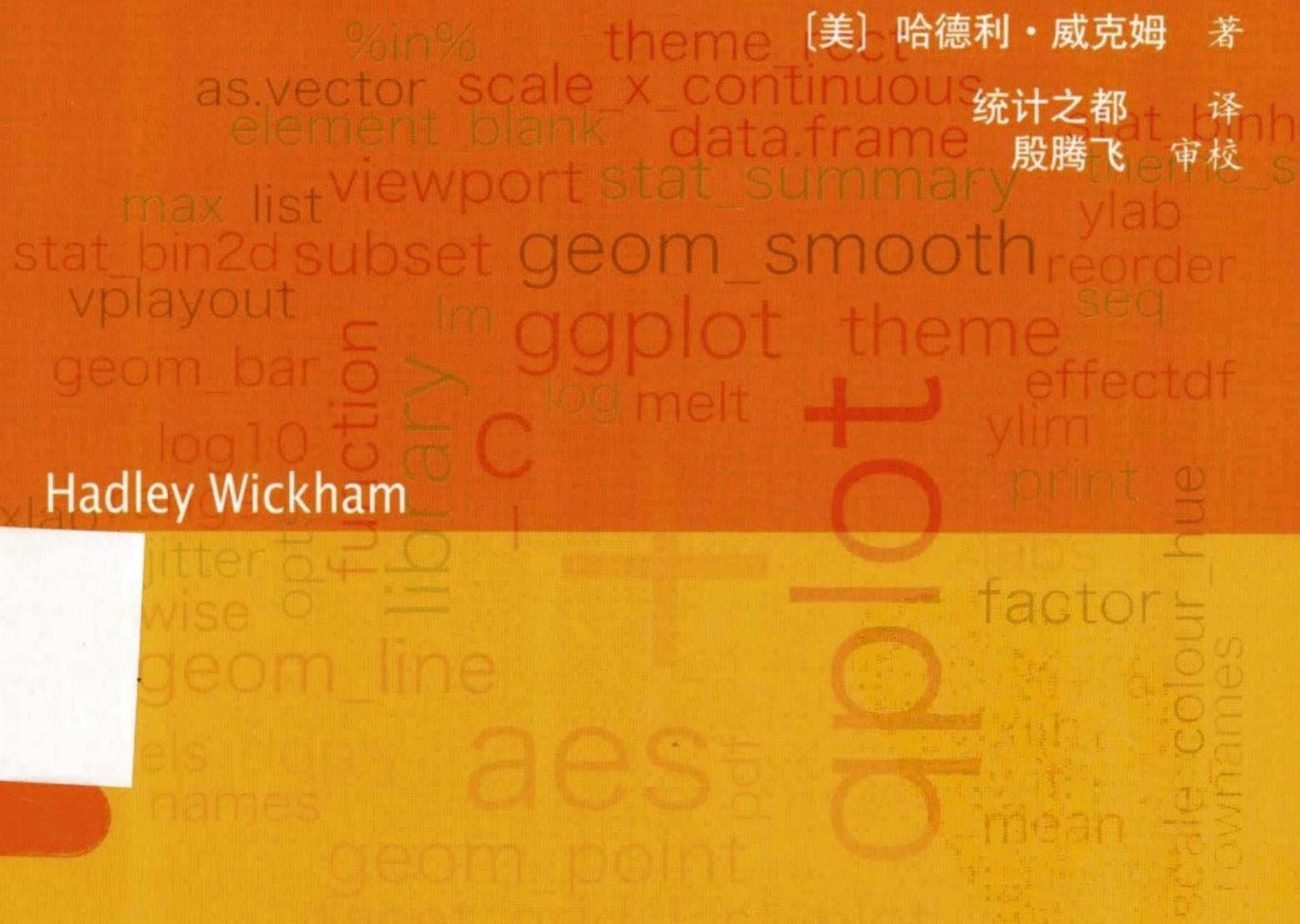


ggplot2: 数据分析与图形艺术

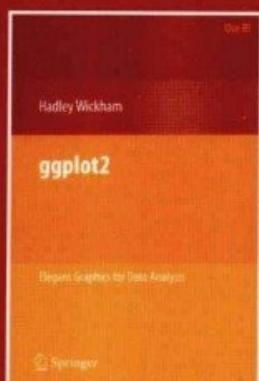
ggplot2. Elegant Graphics for Data Analysis

Hadley Wickham [美] 哈德利·威克姆 著
统计之都 殷腾飞 译
审校

Hadley Wickham



西安交通大学出版社
XI'AN JIAOTONG UNIVERSITY PRESS



玩转数据可视化的瑞士军刀

本书介绍的ggplot2是R中新颖的数据可视化包，功能强大、灵活便捷，其灵感源自Leland Wilkison的《图形的语法》一书。使用ggplot2可以轻松实现：

- 高质量图形的绘制，自动化添加图例。
- 叠加来自不同数据源的多个图层（点、线、地图、瓦片图、箱线图等），自适应通用标度。
- 利用R强大的建模功能添加平滑曲线，如loess、线性模型、广义可加模型和稳健回归。
- 保存任意ggplot2图形，方便修改或重复使用。
- 制作主题，满足内部定制或杂志风格的需求，便捷地应用到多幅图形上。
- 从视觉角度上审视你的图形，斟酌每一部分数据如何呈现在最终图形上。

如果你想将枯燥的数据转化为生动、形象的图片，本书绝对大有裨益。你需要了解R的基本知识（比如如何将数据导入到R），不过ggplot2是专门为绘图量身打造的“迷你语言”，在本书中你就可以学到你需要的所有知识。在阅读完本书之后，你就可以针对你的问题绘制出精确定制的图形，并且发现将脑海中的图片绘制在屏幕上已经变得轻而易举了。

上架建议：程序语言/统计软件/R

ISBN 978-7-5605-4969

A standard barcode is shown, representing the ISBN 978-7-5605-4969.

9 787560 549699 >

责任编辑 李颖 封面设计 阎亮
版式设计 魏太云 特约设计 李苗

定价：46.00元

 Springer

R 语言应用系列

ggplot2. Elegant Graphics for Data Analysis
ggplot2:数据分析与图形艺术

[美] 哈德利·威克姆 著
Hadley Wickham

统计之都 译
殷腾飞 审校

西安交通大学出版社
Xi'an Jiaotong University Press



Translation from the English language edition:
ggplot2: Elegant Graphics for Data Analysis by Hadley Wickham
Copyright © 2009 Springer New York
Springer New York is a part of Springer Science+Business Media
All Rights Reserved

本书中文简体字版由施普林格科学与商业传媒公司授权西安交通大学出版社独家出版发行。
未经出版者预先书面许可，不得以任何方式复制或发行本书的任何部分。

陕西省版权局著作权合同登记号 图字 25 - 2012 - 216 号

图书在版编目(CIP)数据

ggplot2: 数据分析与图形艺术 / (美) 威克姆
(Wickham, H.) 著; 统计之都译. — 西安: 西安
交通大学出版社, 2013. 5
书名原文: Ggplot2. elegant graphics for data analysis
ISBN 978 - 7 - 5605 - 4969 - 9

I. ①g… II. ①威… ②统… III. ①统计分析-应用
软件 IV. ①C819

中国版本图书馆 CIP 数据核字(2013)第 006344 号

书 名 ggplot2: 数据分析与图形艺术
著 者 (美) 哈德利·威克姆
译 者 统计之都
审 校 殷腾飞

出版发行 西安交通大学出版社
(西安市兴庆南路 10 号 邮政编码 710049)
网 址 <http://www.xjtupress.com>
电 话 (029)82668357 82667874(发行中心)
(029)82668315 82669096(总编办)
传 真 (029)82668280
印 刷 陕西宝石兰印务有限责任公司

开 本 720mm×1000mm 1/16 印张 15.5
印 数 0001~3000 字数 253 千字
版次印次 2013 年 5 月第 1 版 2013 年 5 月第 1 次印刷
书 号 ISBN 978 - 7 - 5605 - 4969 - 9/C · 107
定 价 46.00 元

读者购书、书店添货、如发现印装质量问题,请与本社发行中心联系、调换。
订购热线:(029)82665248 (029)82665249

投稿热线:(029)82665397

读者信箱:banquan1809@126.com

版权所有 侵权必究

中译本序

每当我们看到一个新的软件，第一反应会是：为什么又要发明一个新软件？`ggplot2` 是 R 世界里相对还比较年轻的一个包，在它之前，官方 R 已经有自己的基础图形系统 (`graphics` 包) 和网格图形系统 (`grid` 包)，并且 Deepayan Sarkar 也开发了 `lattice` 包，看起来 R 的世界对图形的支持已经足够强大了。那么我们不禁要问，为什么还要发明一套新的系统？

设计理念

打个比方，想想我们小时候怎样学中文的。最开始的时候我们要识字，不认识字就没法阅读和写作，但我们并不是一直按照一个个汉字学习的，而是通过句子和具体的场景故事学习的。为什么不在小学时背六年字典呢？那样可能认识所有的汉字。原因很简单，光有单字，我们不会说话，也无法阅读和写作。我们缺的是什么？答案是对文字的组织能力，或者说语法。

R 的基础图形系统基本上是一个“纸笔模型”，即：一块画布摆在面前，你可以在这里画几个点，在那里画几条线，指哪儿画哪儿。后来 `lattice` 包的出现稍微改善了这种情况，你可以说，我要画散点图或直方图，并且按照某个分类变量给图中的元素上色，此时数据才在画图中扮演了一定的中心角色，我们不用去想具体这个点要用什么颜色（颜色会根据变量自动生成）。然而，`lattice` 继承了 R 语言的一个糟糕特征，就是参数设置铺天盖地，足以让人窒息，光是一份 `xypot()` 函数的帮助文档，恐怕就够我们消磨一天时间了，更重要的是，`lattice` 仍然面向特定的统计图形，像基础图形系统一样，有直方图、箱线图、条形图等等，它没有一套可以让数据分析者说话的语法。

那么数据分析者是怎样说话的呢？他们从来不会说这条线用 #FE09BE 颜色，

那个点用三角形状，他们只会说，把图中的线用数据中的职业类型变量上色，或图中点的形状对应性别变量。有时候他们画了一幅散点图，但马上他们发现这幅图太拥挤，最好是能具体看一下里面不同收入阶层的特征，所以他们会说，把这幅图拆成七幅小图，每幅图对应一个收入阶层。然后发现散点图的趋势不明显，最好加上回归直线，看看回归模型反映的趋势是什么，或者发现图中离群点太多，最好做一下对数变换，减少大数值对图形的主导性。

从始至终，数据分析者都在数据层面上思考问题，而不是拿着水彩笔和调色板在那里一笔一划作图，而计算机程序员则倾向于画点画线。Leland Wilkinson 的著作在理论上改善了这种状况，他提出了一套图形语法，让我们在考虑如何构建一幅图形的时候不再陷在具体的图形元素里面，而是把图形拆分为一些互相独立并且可以自由组合的成分。这套语法提出来之后他自己也做了一套软件，但显然这套软件没有被广泛采用；幸运的是，Hadley Wickham 在 R 语言中把这套想法巧妙地实现了。

为了说明这种语法的思想，我们考虑图形中的一个成分：坐标系。常见的坐标系有两种：笛卡尔坐标系和极坐标系。在语法中，它们属于一个成分，可自由拆卸替换。笛卡尔坐标系下的条形图实际上可以对应极坐标系下的饼图，因为条形图的高可以对应饼图的角度，本质上没什么区别。因此在 ggplot2 中，从一幅条形图过渡到饼图，只需要加极少量的代码，把坐标系换一下就可以了。如果我们用纸笔模型，则可以想象，这完全是不同的两幅图，一幅图里面要画的是矩形，另一幅图要画扇形。

更多的细节在本书中会介绍，这里我们只是简略说明用语法画图对用纸笔画图来说在思维上的优越性；前者是说话，后者是说字。

发展历程

ggplot2 是 Hadley 在爱荷华州立大学博士期间的作品，也是他博士论文的主题之一，实际上 ggplot2 还有个前身 `ggplot`，但后来废弃了，某种程度上这也是 Hadley 写软件的特征，熟悉他的人就知道这不是他第一个“2”版本的包了（还有 `reshape2`）。带 2 的包和原来的包在语法上会有很大的改动，基本上不兼容。尽管如此，他的 R 代码风格在 R 社区可谓独树一帜，尤其是他的代码结构很好，可读性很高，ggplot2 是 R 代码抽象的一个杰作。读者若感兴趣，可以在 GitHub 网站上浏览他的包：<https://github.com/hadley>。在用法方面，

ggplot2 也开创了一种奇特而绝妙的语法，那就是加号：一幅图形从背后的设计来说，是若干图形语法的叠加，从外在的代码来看，也是若干 R 对象的相加。这一点精妙尽管只是 ggplot2 系统的很小一部分，但我个人认为没有任何程序语言可比拟，它对作为泛型函数的加号的扩展只能用两个字形容：绝了。

至 2013 年 2 月 26 日，ggplot2 的邮件列表 (<http://groups.google.com/group/ggplot2>) 订阅成员已达 3394 人，邮件总数为 15185 封，已经成为一个丰富、活跃的用户社区。未来 ggplot2 的发展也将越来越依赖于用户的贡献，这也是很多开源软件最终的走向。

关于版本更新

原书面世之时，ggplot2 的版本号是 0.8.3，译者开始翻译此书时是 0.9.0 版本；该版本较之 0.8.3，内部做了一些大改动。此后，ggplot2 频繁升级，目前版本号是 0.9.3，当然这也给本书的翻译过程带来了相当大的麻烦。因为译者不但要修正原书中大量过时的代码、重新画图，还要修正过时的理念，以及处理数次版本更新的影响。所幸，在翻译过程中，译者得到了本书审校殷腾飞博士、ggplot2 开发者 Hadley Wickham 和 Wistong Chang 的大力帮助。

如果你是老用户，那么可能需要阅读下面的小节。之后 ggplot2 有过多次更新，尤其是 0.9.0 之后，ggplot2 的绘图速度和帮助文档有了质的飞跃。关于 0.9.0 的更新，读者可以从 <https://github.com/downloads/hadley/ggplot2/guide-col.pdf> 下载一份详细的说明文档，但原文档比较长，而且有些内部更新问题我们也不一定需要了解，因此这里给一段概述。

- ggplot2 的帮助文档大大扩充了，过去头疼的问题之一就是一个函数里面不知道有哪些可能的参数，例如 theme() 函数，现在已经有了详细说明。
- 新增图例向导函数 guide_legend() 和 guide_colorbar()，前者可以用来指导图例的排版，例如可以安排图例中元素排为 n 行 m 列；后者增强了连续变量图例的展示，例如当我们把颜色映射到一个连续变量上时，过去生成的图例是离散的，现在可以用这个函数生成连续颜色的图例（渐变色）。
- 新增几何对象函数 geom_map()(让地图语法变得更简单)，geom_raster

()(更高的`geom_tile()`)，`geom_dotplot()`(一维点图，展示变量密度分布) 和 `geom_violin()`(小提琴，实为密度曲线)。

- 新增统计变换函数 `stat_summary2d()`(在二维网格上计算数据密度)，`stat_summary_hex()`(在六边形“蜂巢”上计算数据密度)，`stat_bindot()`(一维点图密度)，`stat_ydensity()`(密度曲线，用于小提琴图)。
- `facet_grid()` 支持 x 轴和 y 轴其中一者可以有自由的刻度(根据数据范围而定)，以往要么所有切片使用同样的坐标轴刻度，要么所有都自由。
- `geom_boxplot()` 开始支持画箱线图的凹槽(notch)，就像 R 基础图形系统中的 `boxplot()` 函数。
- 新增函数 `ggmissing()` 用来展示缺失值的分布，`ggorder()` 按照数据观察顺序先后画折线图，`ggstructure()` 展示数据热图。

另外此次更新涉及到一些函数参数名称的变化，如果旧代码在这个版本中报错说有未使用的参数，那么用户需要再次查看帮助文档，确保输入的参数在函数中存在。在所有这些表面的更新背后，实际上 `ggplot2` 很大程度上被重写了，例如开始使用 R 自带的 S3 泛型函数设计，以及将过去 `ggplot2` 的功能继续模块化为一些独立的包，一个典型的例子就是标度部分的功能被抽象到 `scales` 包中，从数据映射到颜色、大小等外观属性可以由这个包直接完成。这种拆分也使得其他开发者可使用过去 `ggplot2` 内部的一些功能函数。

0.9.1 版本主要解决了 0.9.0 版本中的一些漏洞。`ggplot2` 在 2012 年 9 月 4 日发布了新的版本 0.9.2，其中一些特性和更新有必要提及：

- 采用了全新的主题(theme) 系统，`opts()` 函数已被标记为“不推荐使用”(deprecated)，将在未来版本中被取消，取而代之的是 `theme()` 函数，主题元素(theme element)由属性列表构成，支持继承，主题之间可以直接进行合并等操作。详情参见 wiki 页面：<https://github.com/wch/ggplot2/wiki/New-theme-system>。
- 依赖于新的 `gridExtra` 包。用来更方便地调整修改 `ggplot2` 图形中的图元，`ggplotGrob()` 会返回一个 `grid` 类，这个对象可以利用 `gridExtra` 包中提供的函数和接口进行操作。
- 所有“模板”类型的图形函数，比如 `plotmatrix()`, `ggorder()` 等等，已被标记为“不推荐使用”(deprecated)，将在未来版本中取消。

在本书出版之际，`ggplot2` 更新到了版本 0.9.3，修复了 0.9.2 的一些漏洞，其主要更新包括

- 不再支持 `plotmatrix()` 函数。
- `geom_polygon()` 提速，比如世界地图的绘制快了 12 倍左右。
- 新增部分主题，比如 `theme_minimal()`, `theme_classic()`。

本书的所有代码和图片都是针对新版本 0.9.3 的，在内容方面也根据版本更新对原文做了适当的增删填补，以满足读者的需求。

本书把影响正文阅读的彩图集中放在附录后面，读者可以随时翻阅。

致谢

在听说我们翻译完这本书之后，本书原著 Hadley 很高兴，给我们发邮件说：

I am excited and honoured to have my book translated to Chinese. `ggplot2` has become far more popular than I ever imagined, and I'm excited that this translation will allow many more people to learn `ggplot2`. I'm very grateful that Yihui and his team of translators (Nan Xiao, Tao Gao, Yixuan Qiu, Weicheng Zhu, Taiyun Wei and Lanfeng Pan) made this possible.

One of the biggest improvements to `ggplot2` since the book was first written is the `ggplot2` mailing list. This is a very friendly environment where you can get help with your visualisations, and improve your own knowledge of `ggplot2` by helping others solve their problems. I'd strongly encourage you to join the mailing list, even if you think your English is not very good — we are very friendly people.

我们感谢这本书的译者，包括邱怡轩（第 1~2 章）、主伟呈（第 3~4 章）、肖楠（第 5~6 章）、高涛（第 7~8 章）、潘岚锋（第 9 章）、魏太云（第 10 章、附录以及翻译过程的协调安排和全书的 L^AT_EX 排版工作）。所有译者均来自于统计之都 (<http://cos.name>)。

爱荷华州立大学的殷腾飞博士、中国人民大学统计学院的孟生旺教授、浙江大学的张政同学通读了译稿，提出了很多有用的建议，殷腾飞博士还提供了大多数新版本中的解决方案，并担任本书的审校。肖凯老师和余光创博士分别对第 1~4 章、第 8~10 章以及附录提出了很多修改意见，此外，中国人民大学的陈妍、李晓矛、谢漫锜三位同学、中国再保险公司的李皞先生、百度公司的韩帅先生、eBay 公司的陈丽云女士、Mango Solutions 公司的李舰先生、京东商城的刘思喆先生、首钢总公司的邓一硕先生、新华社的陈堰平先生在此书的翻译过程中也曾提过不少宝贵的建议，在此一并表示感谢。

为了更好地服务社区，我们还建立了翻译主页：<https://github.com/cosname/ggplot2-translation>，读者可以在这里得到最新的勘误和书中的代码，也可以随时提出任何问题。

谢益辉

2013 年 2 月 26 日



即将出版

数据科学中的 R 语言——基础框架和行业应用

李 舰 肖 凯 著



本书是一本 R 语言实战类书籍,目标群体为缺乏编程或者统计基础,但希望能从零开始深入地理解并应用 R 语言的读者。全书分为编程篇、模型篇和应用篇,从 R 的语言特性和分析方法讲起,帮助读者快速入门,然后循序渐进地使读者跟随书中的例子进入到进阶训练,最后应用到实际的案例中。

本书的特点在于行业应用的真实案例。包含了大量从传统的统计分析领域如新药研发、金融分析到当前最热门的大数据、社交网络等应用的例子。作者把从业以来积累的 R 语言在各行业中的应用案例第一次公开出版奉献给读者。书中所有的案例和代码都会做成 R 包发布在 CRAN 上,供读者进行学习和重用。

2013 年 12 月出版 380 页 R 语言应用系列

R 语言的科学编程与仿真

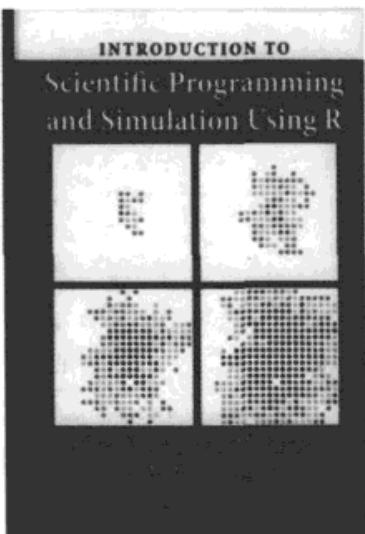
Owen Jones Robert Maillardet Andrew Robinson 著

王 亮 周丙常(西北工业大学)

王 亮(西安电子科技大学) 译

这本书主要介绍了科学编程与随机建模的一些技巧。旨在整合科学编程和概率论,尤其是通过数值模拟实现现代概率统计理论的应用。随书附带的 spuRs 包含了大部分提到的程序,可供读者调试使用。

不同于大部分 R 的指导书,本书不仅介绍了统计方法的应用,还介绍了如何将算法转化为代码。这一点对于那些不仅希望使用代码,而且想要设计代码的人是很有用的。



2013 年 11 月出版 400 页 R 语言应用系列

目录

中译本序	1
目录	i
第 1 章 简介	1
1.1 欢迎来到 <code>ggplot2</code> 的世界	1
1.2 其他资源	2
1.3 什么是图形的语法?	3
1.4 <code>ggplot2</code> 与 R 中其他软件包的对比	4
1.5 关于本书	6
1.6 安装	7
1.7 致谢	7
第 2 章 从 <code>qplot</code> 开始入门	8
2.1 简介	8
2.2 数据集	9
2.3 基本用法	10
2.4 颜色、大小、形状和其他图形属性	11
2.5 几何对象	13
2.5.1 向图中添加平滑曲线	14
2.5.2 箱线图和扰动点图	17
2.5.3 直方图和密度曲线图	18
2.5.4 条形图	20
2.5.5 时间序列中的线条图和路径图	21

2.6 分面	23
2.7 其他选项	25
2.8 与 plot 函数的区别	26
第 3 章 语法突破	28
3.1 简介	28
3.2 耗油量数据	29
3.3 绘制散点图	30
3.4 更复杂的图形示例	35
3.5 图层语法的组件	37
3.5.1 图层	38
3.5.2 标度	38
3.5.3 坐标系	39
3.5.4 分面	40
3.6 数据结构	40
第 4 章 用图层构建图像	42
4.1 简介	42
4.2 创建绘图对象	43
4.3 图层	43
4.4 数据	46
4.5 图形属性映射	47
4.5.1 图和图层	48
4.5.2 设定和映射	50
4.5.3 分组	51
4.5.4 匹配图形属性和图形对象	54
4.6 几何对象	57
4.7 统计变换	60
4.8 位置调整	62
4.9 整合	63
4.9.1 结合几何对象和统计变换	63
4.9.2 显示已计算过的统计量	65
4.9.3 改变图形属性和数据集	65

第 5 章 工具箱	68
5.1 简介	68
5.2 图层叠加的总体策略	69
5.3 基本图形类型	70
5.4 展示数据分布	72
5.5 处理遮盖绘制问题	77
5.6 曲面图	82
5.7 绘制地图	82
5.8 揭示不确定性	85
5.9 统计摘要	89
5.9.1 单独的摘要计算函数	89
5.9.2 统一的摘要计算函数	90
5.10 添加图形注解	91
5.11 含权数据	95
第 6 章 标度、坐标轴和图例	98
6.1 简介	98
6.2 标度的工作原理	99
6.3 用法	100
6.4 标度详解	103
6.4.1 通用参数	103
6.4.2 位置标度	105
6.4.3 颜色标度	110
6.4.4 手动离散型标度	115
6.4.5 同一型标度	119
6.5 图例和坐标轴	119
6.6 更多资源	122
第 7 章 定位	123
7.1 简介	123
7.2 分面	123
7.2.1 网格分面	124
7.2.2 封装分面	129

7.2.3 标度控制	130
7.2.4 分面变量缺失	133
7.2.5 分组与分面	133
7.2.6 并列与分面	135
7.2.7 连续型变量	136
7.3 坐标系	139
7.3.1 变换	139
7.3.2 统计量	141
7.3.3 笛卡尔坐标系	141
7.3.4 非笛卡尔坐标系	145
第 8 章 精雕细琢	147
8.1 主题	147
8.1.1 内置主题	148
8.1.2 主题元素和元素函数	150
8.2 自定义标度和几何对象	156
8.2.1 标度 ^①	156
8.2.2 几何对象和统计变换	156
8.3 存储输出	157
8.4 一页多图	159
8.4.1 子图	160
8.4.2 矩形网格	162
第 9 章 数据操作	164
9.1 <code>plyr</code> 包简介	164
9.1.1 拟合多个模型	168
9.2 把数据化“宽”为“长”	171
9.2.1 多重时间序列	172
9.2.2 平行坐标图	175
9.3 <code>ggplot()</code> 方法	178
9.3.1 线性模型	179
9.3.2 编写自己的方法	182

^①本节为适应 0.9.0 及之后版本而重写

第 10 章 减少重复性工作	184
10.1 简介	184
10.2 迭代	184
10.3 绘图模板	185
10.4 绘图函数	188
附录 A 不同语法间的转换	190
A.1 简介	190
A.2 在 qplot 和 ggplot 间转换	190
A.2.1 图形属性	191
A.2.2 图层	191
A.2.3 标度和坐标轴	192
A.2.4 绘图选项	192
A.3 基础图形系统	192
A.3.1 高级绘图	193
A.3.2 低级绘图	194
A.3.3 图例、坐标轴和网格线	195
A.3.4 调色板	195
A.3.5 绘图参数	196
A.4 lattice 图形设备	196
A.5 GPL	198
附录 B 图形属性的定义	200
B.1 颜色	200
B.2 线条类型	200
B.3 形状	202
B.4 大小	202
B.5 对齐方式	202
附录 C 用 grid 操作图形	203
C.1 简介	203
C.2 视图窗口	203
C.3 绘制图形元件	205

C.4 保存工作	206
参考文献	208
主题索引	212
函数索引	216
彩色插图	218



第 1 章 简介

1.1 欢迎来到 `ggplot2` 的世界

`ggplot2` 是一个用来绘制统计图形 (或称为数据图形) 的 R 软件包。与其他大多数的图形软件包不同, `ggplot2` 是由其背后的一套图形语法所支持的。这一语法基于《Grammar of Graphics》(Wilkinson, 2005) 一书, 它由一系列独立的图形部件组成, 并能以许多种不同的方式组合起来。这一点使得 `ggplot2` 的功能非常强大, 因为它不会局限于一些已经定义好的统计图形, 而是可以根据你的需要量身定做。这听起来似乎有些困难, 但实际上只需要掌握一些核心准则以及少许特例, `ggplot2` 还是很容易学习的 (尽管你可能需要花费一些时间来忘掉其他图形工具中一些固有的概念)。

`ggplot2` 可以绘制出很多美观的图形, 同时能避免诸多繁琐的细节, 例如添加图例等。用 `ggplot2` 绘图时, 图形的每个部分可以依次进行构建, 之后还可以进行编辑。`ggplot2` 精心挑选了一系列预设图形, 因此在大部分的情形下你可以快速地绘制出许多高质量的图形。如果在格式上还有额外的需求, 也可以利用 `ggplot2` 中的主题系统来进行定制。这样, 你将无需花费太多时间来调整图形的外观, 而可以更加专注地用图形来展示你的数据。

`ggplot2` 采用了图层的设计方式, 你可以从原始的图层开始, 首先绘制原始数据, 然后不断地添加图形注释和统计汇总结果。这种绘图方式与分析问题中的结构化思维是一致的, 它能缩短你“所思”与“所见”的距离。特别地, `ggplot2` 可以帮助学生锻炼结构化的分析思维, 进而达到专业的水准。

学习图形语法不仅可以帮助你绘制出你已经了解的图形, 甚至还可以启发你创作出更佳的方案。如果没有这一套语法体系, 图形的绘制便失去了理论支持, 这也就是为什么现有的很多图形软件包只是一系列特例的堆积。例如, 在

R 的基础包中，如果你想要设计一个新的图形，那么它一定是由原始的图形元素（如点和线）构成的，而你很难利用已有的图形样例进行组合，以生成新的图形部件。在 `ggplot2` 中，用于创建新图形的表达式是由高级的图形元素组成的，例如对原始数据的展现，以及某些统计变换等。这些图形元素可以很方便地与新的数据集以及其他图形进行结合。

本书对 `ggplot2` 进行了整体的介绍，并配以大量的样例代码和图形，同时解释了 `ggplot2` 背后的语法。与其他许多成熟的系统一样，即使你不了解 `ggplot2` 背后的模型，你也可以用它来完成许多工作。当然，你了解的内容越多，使用 `ggplot2` 时就会越高效。本书假定你已经熟悉了 R 的基本操作，水平大致相当于 Dalgaard 在《Introductory Statistics with R》的第 1 章中描述的那样。同时你也需要了解如何将数据导入 R 中并进行基本的数据操作。如果你还不具备这些能力，则可以参考 Phil Spector 的《Data Manipulation with R》一书。

本书会从零开始介绍 `ggplot2` 的语法，因此读者朋友不用担心目前对该语法一无所知。你将会学习到如何利用 `ggplot2` 来重新绘制那些你已经很熟悉的图形，以及如何运用语法来创建新的图形。甚至，我们还会让你成为这方面的专家，由你自己来构建新的图形部件，以扩展这一套语法。

1.2 其他资源

本书将教给你 `ggplot2` 的基本语法以及如何将它们组织起来，但并不会对每一个函数都进行完整的介绍。此外，`ggplot2` 将会不断地更新和改进，因此，当你想要运用 `ggplot2` 更复杂的技巧时，你将需要额外的帮助文档。

获取底层细节的最佳途径是 `ggplot2` 内置的文档，你可以在线访问 <http://had.co.nz/ggplot2> 或利用 R 中的帮助系统来获取。在线帮助文档的优点是你可以看到所有的示例图形，同时可以更方便地在主题之间进行跳转。

上述网站中还列出了与 `ggplot2` 有关的演讲和文章资料，如果你想亲自进行实践，网站中也提供了培训的相关信息。CRAN 网站 <http://cran.r-project.org/web/packages/ggplot2/> 是另一个有用的资源，这个页面将会链接到 `ggplot2` 的最新发布版本，并列出版本之间改进的内容。如果你经常使用 `ggplot2`，你还可以考虑注册 `ggplot2` 的邮件列表 <http://groups.google.com/group/ggplot2>，它对新用户非常有帮助。

最后，本书的网站 <http://had.co.nz/ggplot2/book> 提供了对本书的更新以及书中所有图形的 pdf 文件、程序代码和必要的数据集。

1.3 什么是图形的语法?

Wilkinson (2005) 创建了一套用来描述所有统计图形深层特性的语法规则，该语法回答了“什么是统计图形”这一问题。Wickham (2010) 提出了基于 Wilkinson 语法的图层图形语法，它将图层作为主要的考察对象，并使其嵌入到 R 中。简而言之，这套语法告诉我们，一张统计图形就是从数据到 **几何对象** (geometric object，缩写为 geom，包括点、线、条形等) 的 **图形属性** (aesthetic attributes，缩写为 aes，包括颜色、形状、大小等) 的一个映射。此外，图形中还可能包含数据的 **统计变换** (statistical transformation，缩写为 stats)，最后绘制在某个特定的 **坐标系** (coordinate system，缩写为 coord) 中，而 **分面** (facet，指将绘图窗口划分为若干个子窗口) 则可以用来生成数据不同子集的图形。总而言之，一张统计图形就是由上述这些独立的图形部件所组成的。

随着本书内容的推进，我们将会愈加详细地解释完整的语法。以下是对这些图形部件的第一次描述，其中的一些术语将会贯穿整本书，它们概括了这些图形部件的基本作用。如果你现在还无法完全理解，请不要担心：你在之后会有很多机会来了解其中的细节，并弄清楚它们是如何组合在一起的。

- 最基础的部分是你想要可视化的**数据** (data) 以及一系列将数据中的变量对应到图形属性的**映射** (mapping)；
- **几何对象** (geom) 代表你在图中实际看到的图形元素，如点、线、多边形等；
- **统计变换** (stats) 是对数据进行的某种汇总。例如，将数据分组计数以创建直方图，或将一个二维的关系用线性模型进行解释。统计变换是可选的，但通常非常有用；
- **标度** (scale) 的作用是将数据的取值映射到图形空间，例如用颜色、大小或形状来表示不同的取值。展现标度的常见做法是绘制图例和坐标轴——它们实际上是从图形到数据的一个映射，使读者可以从图形中读取原始的数据；
- **坐标系** (coord) 描述了数据是如何映射到图形所在的平面的，它同时提供

了看图所需的坐标轴和网格线。我们通常使用的是笛卡尔坐标系，但也可以将其变换为其他类型，如极坐标和地图投影；

- **分面 (facet)**描述了如何将数据分解为各个子集，以及如何对子集作图并联合进行展示。分面也叫做条件作图或网格作图。

在此有必要说明一下图形语法的一些局限：

- 对于你感兴趣的问题，它没有指出你应该采取哪一种图形来进行展示。本书旨在提出一种合理的绘制数据图形的过程，因此讨论的重点是如何画出你想要的图形，而不是告诉你应该用哪一种图形。如果你对后者感兴趣，则可以参阅 Chambers et al. (1983); Cleveland (1993); Robbins (2004); Tukey (1977)；
- 有些讽刺的是，图形语法并没有规定一张图形的外观应该是怎样的。图形应该如何进行展示，例如字体大小和背景颜色的选取，并不是由语法规规定的。然而在实际操作中，一个理想的作品系统应该要描述这些内容，因此 `ggplot2` 采用了它自身的主题系统来实现这些功能。类似地，语法也没有说明如何创建一张吸引眼球的图形。尽管 `ggplot2` 的默认设置是经过精心设计的，但你也许还需要参阅其他的资料来绘制出一张魅力四射的图形：Tufte (1990, 1997, 2001, 2006)；
- 它没有描述交互式的操作：图形语法仅仅描述了静态的图形，因此对它而言，图形在电脑屏幕上或是在纸张上没有太大的区别。`ggplot2` 只能创建静态图形，如果你想绘制动态或交互式的图形，你就需要查阅其他的文献。Cook and Swayne (2007) 提供了一份对交互式图形软件包 GGobi 完善的介绍。GGobi 可以通过 `rggobi` 软件包与 R 进行连接 (Wickham et al., 2008)。

1.4 `ggplot2` 与 R 中其他软件包的对比

在 R 中有许多其他的图形系统：基础图形系统、网格图形系统和 `trellis/lattice` 图形系统等。`ggplot2` 与它们的不同体现在以下几个方面：

- 基础图形系统是由 Ross Ihaka 编写的，这基于他实现 S 语言图形驱动的经验，同时参照了 Chambers et al. (1983)。基础图形系统采用的是“纸一笔”的模型：你只能在图形的最顶端进行绘画，而不能修改或删除已有的

内容；除了已经呈现在屏幕上的图像之外，图形没有其他（用户可控）的表达形式，如层次、结构、元素类型等。基础图形系统既能绘制基本的图形元素，又提供了绘制完整图形的工具，其中大部分的函数运行速度很快，但使用范围有限。如果你曾经画过单张的散点图、直方图或一系列箱线图，那么你所用的可能就是基础图形系统；

- 网格图形系统 (`grid`) 的开发始于 2000 年，是一个更加丰富的绘制基本图形元素的系统。它由 Paul Murrell 开发，是从其博士的工作衍生出来的 (Murrell, 1998)。网格系统中的图形对象可以与屏幕上的图像独立存在，并且可以反复修改，其中一系列的视图区（每一个视图区都有其自己的坐标系）可以方便地对图形进行复杂的布局。网格系统只能绘制基本的图形元素，而没有提供创建统计图形的工具；
- `lattice` 软件包 (Sarkar, 2008b) 由 Deepayan Sarkar 开发，它利用网格图形系统实现了 Cleveland (1985, 1993) 中提出的 `lattice` 图形系统，这是对基础绘图系统的一大改进。你可以利用 `lattice` 软件包方便地进行条件作图，而且其中的一些作图细节（例如图例）会自动地完成。然而，`lattice` 图形系统缺少一个完整的模型，这使得它很难进行扩展。`lattice` 图形系统在 Sarkar (2008a) 中有详细的讨论；
- `ggplot2` 于 2005 年开始出现，它试图吸取基础绘图系统和 `lattice` 绘图系统的优点，并利用一个强大的模型来对这两个系统进行改进。这一模型基于之前所述的一系列准则，能够创建任何类型的统计图形。`ggplot2` 背后的模型使得用户可以用紧凑的语法轻松地绘制出许多种图形，而且其中独立的图形部件使得图形的可扩展性很强。与 `lattice` 类似，`ggplot2` 利用了网格图形系统来进行作图，这意味着你可以在底层对图形的外观进行控制。

许多其他的 R 软件包，例如 `vcd` (Meyer et al., 2006)，`plotrix` (Lemon et al., 2008) 和 `gplots` (Warnes, 2007)，它们开发了一些针对特殊需要的图形，但目前还没有其他的软件包能够提供绘制统计图形的统一框架。你可以在以下网页中找到一份完整的 R 中图形软件包的列表，<http://cran.r-project.org/web/views/Graphics.html>。

1.5 关于本书

本书第 2 章介绍了如何利用 `qplot` 函数来快速入门，就好像用 `plot` 函数来绘制图形一样。这一章介绍了 `ggplot2` 中一些重要的概念：几何对象、图形映射以及分面。

尽管 `qplot` 是一条快速入门的途径，但它还没有展现图形语法的完整功能。本书第 3 章介绍了 `ggplot2` 背后的图层图形语法，第 4 章阐释了其中的理论，并演示了如何在图形中添加图层，以及如何完全控制其中的几何对象和统计变换。第 5 章描述了如何将几何对象和统计变换进行组合，从而解决特定的作图问题。

理解标度是如何工作的对于优化图形的感官属性非常重要。通过自定义标度，你可以精确地控制图形的外观，以便于更好地表达你的意图。本书第 6 章介绍了一些常用的标度，并对如何调整它们的参数以及如何控制坐标轴和图例的外观做了简要说明。

坐标系和分面可以控制图形元素的位置，这些内容将在第 7 章进行讲解。分面是一个非常强大的图形工具，它可以帮助你快速地比较数据的不同子集。在实际应用中一般不需要改变图形的坐标系，但对于某些特定类型的数据，坐标系的选择会显得非常重要。

为了进一步优化你的图形以达到较高的质量，你将需要本书第 8 章中介绍到的工具。在那里你将学习到如何控制 `ggplot2` 的主题系统，如何改变几何对象、统计变换和标度的默认取值，如何将图形保存到硬盘，以及如何在一页纸中放置多张图等。

本书的最后两章讨论了关于数据结构和代码冗余的高级话题。第 9 章讨论了将你的数据转换为 `ggplot2` 所需格式的一些技巧，并介绍了比作图命令更为高级的数据汇总和处理的工具。你还将学习到 `ggplot2` 在对其他类型的对象进行可视化时背后的哲学，以及你如何用自己的方法来对 `ggplot2` 进行扩展。

代码冗余是对代码可扩展性的一大阻碍，它会降低你应对变化的能力。本书第 10 章涵盖了三个减少代码冗余的有用技巧：逐步修改、绘图模板和绘图函数。

本书的三个附录收录了一些额外的信息。附录 A 展示了如何将基础绘图系统、`lattice` 绘图系统和 Wilkinson 的 GPL(Graphics Production Library) 的语法转变为 `ggplot2` 的语法。附录 B 介绍了如何手动指定颜色、形状、线条类型和大小。附录 C 描述了 `ggplot2` 中网格图形对象的高级组织方式，以及其中视

图区的布局。如果你对网格图形系统比较熟悉，并希望修改图形背后的对象，那么这些信息将是非常有帮助的。

1.6 安装

要使用 `ggplot2`，必须先安装它。首先确保你使用的是较新版本的 R 软件（2.8 版本及以上），可以从 <http://r-project.org> 进行下载。之后，运行如下的命令来下载和安装 `ggplot2` 软件包。

```
1 | install.packages("ggplot2")
```

`ggplot2` 并不是完美的，因此在实际工作中它可能会出现一些问题。如果产生了错误，请发送邮件至 `hadley@rice.edu`，并最好提供一个可重复的例子，告诉我哪里出现了问题。你提供的信息越多，问题越有可能最终得到解决。

1.7 致谢

有许多人为本书的出版做出了贡献，他们在本书的整体结构、语法拼写和程序运行等方面提出了宝贵的意见。特别地，我要感谢 Leland Wilkinson，他和我的讨论以及提出的意见加深了我对语法的认识；感谢 Gabor Grothendieck 在早期提出的许多建设性的意见；感谢 Heike Hofmann 和 Di Cook，他们是这一领域出色的教授；感谢 Charlotte Wickham；感谢爱荷华州立大学 stat480 和 stat503 班级的学生们，他们在 `ggplot2` 还不成熟的时候就进行了测试；感谢 Debby Swayne 提供的许多有益的反馈和建议；感谢 Bob Muenchen、Reinhold Kliegl、Philipp Pagel、Richard Stahlhut、Baptiste Auguie、Jean-Olivier Irisson、Thierry Onkelinx 以及其他读过本书初稿并给予了反馈的人；最后，还要感谢 R-help 和 `ggplot2` 邮件列表的成员，他们提供了许多有趣且富有挑战性的图形问题，这些问题都促进了本书的完成。

第 2 章 从 `qplot` 开始入门

2.1 简介

在本章中，`qplot()` 将是你学习的第一个 `ggplot2` 作图函数。`qplot` 的意思是快速作图 (quick plot)，利用它你可以很方便地创建各种复杂的图形。有时在其他图形系统中需要好几行代码才能解决的作图问题，用 `qplot()` 只需要一行就能完成。`qplot()` 之所以有这样的能力，是因为它基于图形语法，这使得你能用简单的、富有表达能力的语句来描述一张图形。在之后的章节中你将学习到完整的语法，而在这里我们先介绍简单的内容，以便你能快速入门。在本章中你将会同样学习到一些贯穿全书的 `ggplot2` 术语。

`qplot()` 被设计得与 `plot()` 很像，因此如果你已经对 R 中的作图比较熟悉，那么用起它来也会很容易。记住，在一次 R 会话中，你可以通过 R 的帮助命令 `?qplot` 来获取 `qplot()` 的完整参数列表。

在本章中你将学习到：

- `qplot()` 的基本用法。如果你已经对 `plot()` 很熟悉，那么这部分内容将很简单 (§ 2.3)；
- 如何将变量映射到图形属性 (如颜色、大小和形状) 之上 (§ 2.4)；
- 如何通过指定不同的几何对象来创建不同类型的图形，以及如何将它们组合在一张图中 (§ 2.5)；
- 分面 (或称为条件作图) 的运用，将数据拆分为子集 (§ 2.6)；
- 如何通过设定基本的选项来调整图形的外观 (§ 2.7)；
- `qplot()` 与 `plot()` 之间一些重要的区别 (§ 2.8)。

2.2 数据集

本章大部分的绘图都将只使用一个数据源，这样你可以更好地熟悉作图的细节，而不需要去熟悉各种不同的数据集。`diamonds` 数据集包含了约 54000 颗钻石的价格和质量的信息，数据已经放在了 `ggplot2` 软件包中。这组数据涵盖了反映钻石质量的四个“C”——克拉重量 (`carat`)、切工 (`cut`)、颜色 (`color`) 和净度 (`clarity`)，以及五个物理指标——深度 (`depth`)、钻面宽度 (`table`)、`x`、`y` 和 `z`，如图 2.1 所示。数据的前 6 行展示在表 2.1 中。

表 2.1 `diamonds` 数据集。变量 `depth`、`table`、`x`、`y` 和 `z` 代表了钻石的各种测量尺度，如图 2.1 所示。

carat	cut	color	clarity	depth	table	price	x	y	z
0.2	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.2	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.2	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.2	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.2	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
0.2	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48

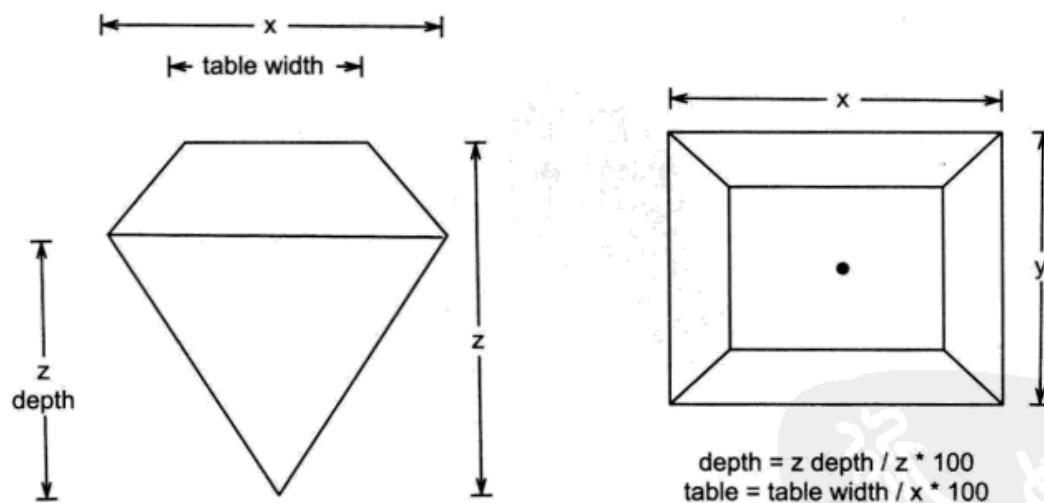


图 2.1 变量 `x`、`y`、`z`、`table` 和 `depth` 的测量示意图。

这个数据集没有经过很好的整理，所以在展示钻石一些有趣的关系时，它会显示出一些数据质量的问题。我们将同时使用另一个数据集，`dsmall`，它是原始数据的一个容量为 100 的随机样本。我们将用这个数据集来进行小数据的作图展示。

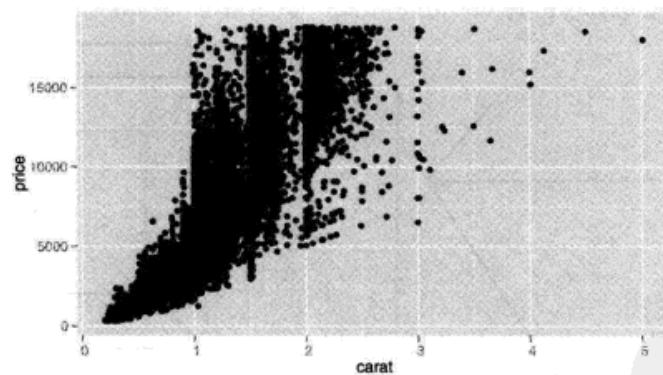
```
1 set.seed(1410) ## 让样本可重复  
2  
3 dsmall <- diamonds[sample(nrow(diamonds), 100), ]
```

2.3 基本用法

与 `plot` 类似，`qplot()` 的前两个参数是 `x` 和 `y`，分别代表图中所画对象的 `x` 坐标和 `y` 坐标。此外，还有一个可选的 `data` 参数，如果进行了指定，那么 `qplot()` 将会首先在该数据框内查找变量名，然后再在 R 的工作空间中进行搜索。本书推荐使用 `data` 参数，因为将相关的数据放置在同一个数据框中是一个良好的习惯。如果你没有指定 `data` 参数，那么 `qplot()` 将会尝试建立一个，但这样做有可能会使程序在错误的地方查找变量。

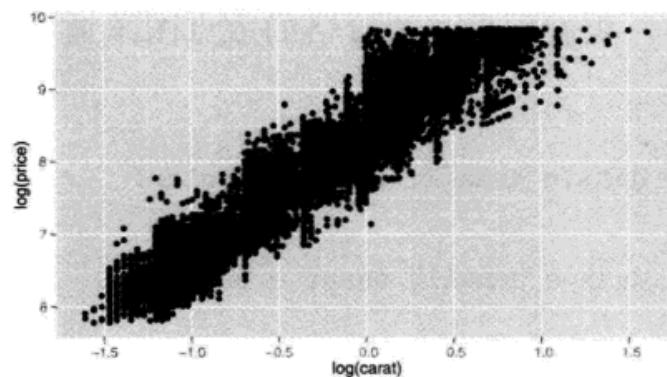
下面是使用 `qplot()` 的一个简单的例子。它绘制了一张散点图，展现了钻石价格和重量之间的关系。

```
1 qplot(carat, price, data = diamonds)
```



这张图显示出了变量之间很强的相关关系，以及一些很明显的异常值，此外，可以看出在竖直方向上有一些有趣的条纹。这种相关关系似乎是指数型的，因此我们应该首先对变量进行一些变换。由于 `qplot()` 支持将变量的函数作为参数，因此我们可以画出 $\log(\text{price})$ 对 $\log(\text{carat})$ 的图形：

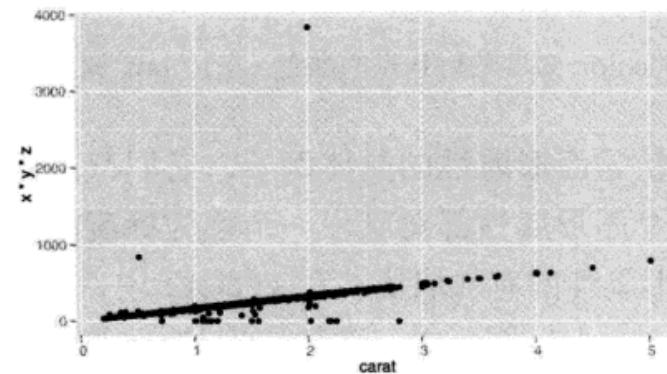
```
1 qplot(log(carat), log(price), data = diamonds)
```



现在这种关系就接近于线性了。然而，由于图中的元素有很大的重叠，所以我们在下结论时需要小心。

函数的参数同样可以是已有变量的某种组合。例如，如果我们将钻石的体积（用 $x \times y \times z$ 近似）和其重量之间的关系感兴趣，那么我们可以这样做：

```
1 qplot(carat, x * y * z, data = diamonds)
```



可以预期的是，钻石的密度（质量除以体积）应该是一个常数，因此体积与重量之间应该是线性的关系。从图中可以看出，大部分的钻石都落在同一条直线上，但依然存在一些大的异常点。

2.4 颜色、大小、形状和其他图形属性

`qplot` 与 `plot` 的第一个大的差别在于它们在给图中的点设定颜色（或大小、形状）时采用了不同的实现方式。在 `plot` 中，用户需要将数据中的一个分类变量（例如，“苹果”、“香蕉”、“桃子”）转换为 `plot` 可以理解的形式（例如，“red”、“yellow”、“green”）。而 `qplot` 可以将这个过程自动完成，并能够自动生成一张图例，用以展示数据取值与图形属性之间的对应关系。这使得向图中添加额外的信息非常简便。

在下一个例子中，我们向重量和价格的散点图添加了颜色和切工的信息，结果展示在图 2.2 中。

```
1 qplot(carat, price, data = dsmall, colour = color)
2
3 qplot(carat, price, data = dsmall, shape = cut)
```

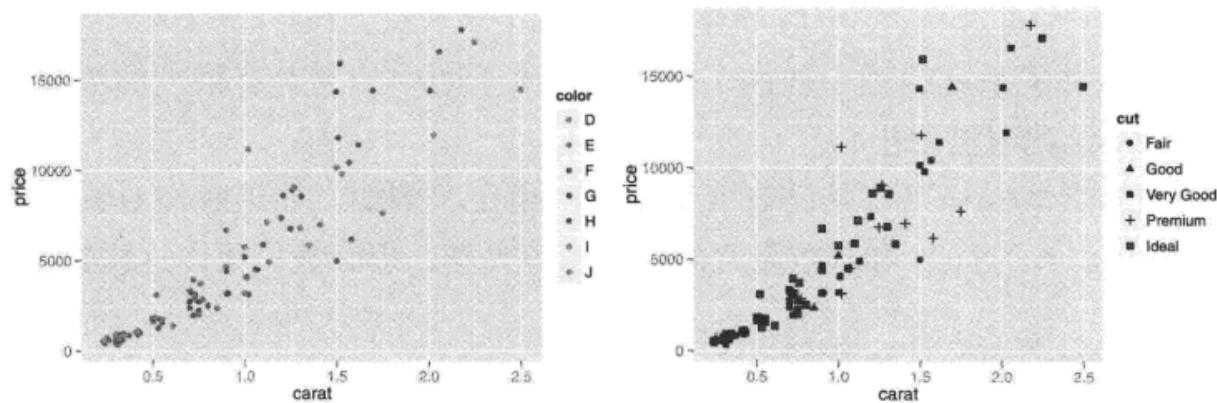


图 2.2 (参见彩图 1) 将 color 变量映射到点的颜色 (左), cut 变量映射到点的形状 (右)。

颜色、大小和形状是图形属性的具体例子，它们都是影响数据如何进行展示的视觉属性。每一个图形属性都对应了一个称为标度的函数，其作用是将数据的取值映射到该图形属性的有效取值。在 ggplot2 中，标度控制了点以及对应的图例的外观。例如，在上述图形中，颜色标度将 J 映射为紫色，将 F 映射为绿色。

你同样可以利用 `I()` 来手动设定图形属性，例如，`colour = I("red")` 或 `size = I(2)`。这与之前解释的映射不同，其中更详细的内容会在 4.5.2 节进行介绍。对于大数据（如本例中的钻石数据）而言，使用半透明的颜色可以有效减轻图形元素重叠的现象。要创建半透明的颜色，你可以使用 `alpha` 图形属性，其取值从 0（完全透明）变动到 1（完全不透明）。通常透明度可以用分数来进行表示，例如 `1/10` 或 `1/20`，其分母表示经过多少次重叠之后颜色将变得不透明。

```
1 qplot(carat, price, data = diamonds, alpha = I(1/10))
2
3 qplot(carat, price, data = diamonds, alpha = I(1/100))
4
5 qplot(carat, price, data = diamonds, alpha = I(1/200))
```

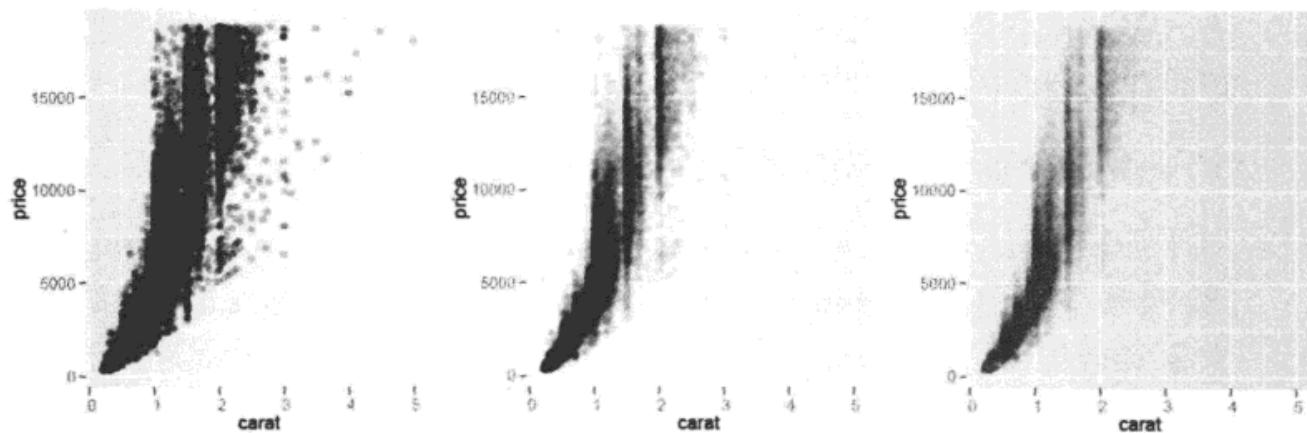


图 2.3 将 `alpha` 值从 $1/10$ (左) 变动到 $1/100$ (中) 再到 $1/200$ (右)，来看大部分的点在哪里进行重叠。

不同类型的变量有不同适用的图形属性。例如，颜色和形状适合于分类变量，而大小适合于连续变量。数据量的大小同样会有影响：如果数据量很大(如上图)，那么不同组的数据之间就很难进行区分。对此，一种可能的解决方案是利用分面，这在 2.6 节会有介绍。

2.5 几何对象

`qplot` 并非只能画散点图，通过改变几何对象(简写为 `geom`)，它几乎可以画出任何一种类型的图形。几何对象描述了应该用何种对象来对数据进行展示，其中有些几何对象关联了相应的统计变换。例如，直方图就相当于分组计数再加上条形的几何对象。这些不同的图形部件将在下一章进行描述，在此，我们介绍最常用的几种，组织的方式是按照数据的维度进行划分。下面这些几何对象适用于考察二维的变量关系：

- `geom = "point"` 可以绘制散点图。这是当你指定了 `x` 和 `y` 参数给 `qplot()` 时默认的设置；
- `geom = "smooth"` 将拟合一条平滑曲线，并将曲线和标准误展示在图中(§ 2.5.1)；
- `geom = "boxplot"` 可以绘制箱线胡须图，用以概括一系列点的分布情况(§ 2.5.2)；
- `geom = "path"` 和 `geom = "line"` 可以在数据点之间绘制连线。这类图传统的作用是探索时间和其他变量之间的关系，但连线同样可以用其他的

方式将数据点连接起来。线条图只能创建从左到右的连线，而路径图则可以是任意的方向（§ 2.5.5）。

对于一维的分布，几何对象的选择是由变量的类型指定的：

- 对于连续变量，`geom = "histogram"` 绘制直方图，`geom = "freqpoly"` 绘制频率多边形，`geom = "density"` 绘制密度曲线（§ 2.5.3）。如果只有 `x` 参数传递给 `qplot()`，那么直方图几何对象就是默认的选择；
- 对于离散变量，`geom = "bar"` 用来绘制条形图（§ 2.5.4）。

2.5.1 向图中添加平滑曲线

如果在散点图中有非常多的数据点，那么数据展示的趋势可能并不明显，在这种情况下你应该在图中添加一条平滑曲线。这可以通过使用 `smooth` 几何对象加以完成，如图 2.4 所示。注意到我们利用了 `c()` 函数来将多个几何对象组成一个向量传递给 `geom`。几何对象会按照指定的顺序进行堆叠。

```
1 qplot(carat, price, data = dsmall, geom = c("point", "smooth"))
2
3 qplot(carat, price, data = diamonds, geom = c("point", "smooth"))
```

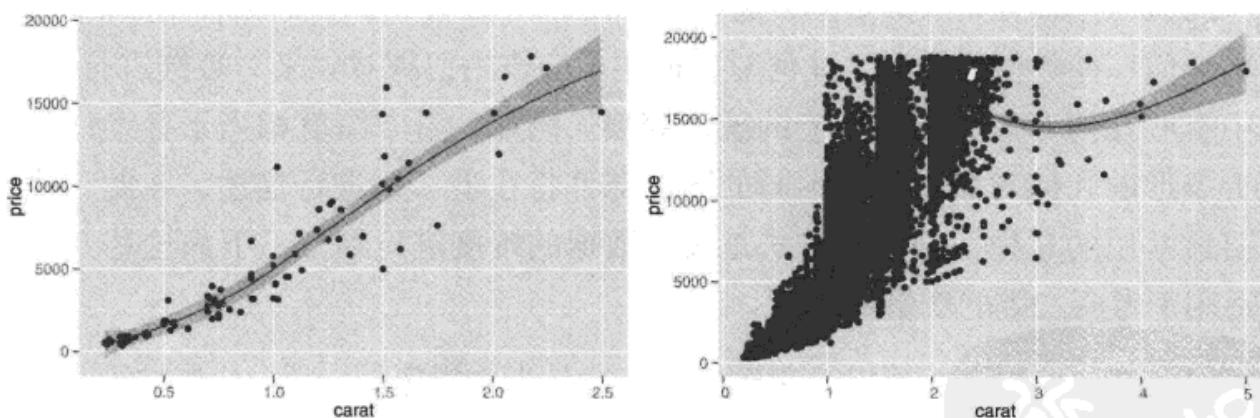


图 2.4 重量与价格的散点图中加入了平滑曲线。左图为 `dsmall` 数据集，右图为完整数据集。

尽管图形元素重叠严重，但我们关于价格和重量之间指数关系的猜想基本上是正确的。由于只有很少的钻石超过了 3 克拉，因此这种关系的不确定性程度也在逐渐加大，这与灰色部分的逐点置信区间的宽度变化是一致的。如果你不想绘制标准误，则可以使用 `se = FALSE`。

利用 `method` 参数你可以选择许多不同的平滑器：

- `method = "loess"`, 当 n 较小时是默认选项, 使用的是局部回归的方法。关于这一算法的更多细节可以查阅帮助 `?loess`。曲线的平滑程度是由 `span` 参数控制的, 其取值范围是从 0 (很不平滑) 到 1 (很平滑), 如图 2.5 所示。

```

1 qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
2   span = 0.2)
3 qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
4   span = 1)

```

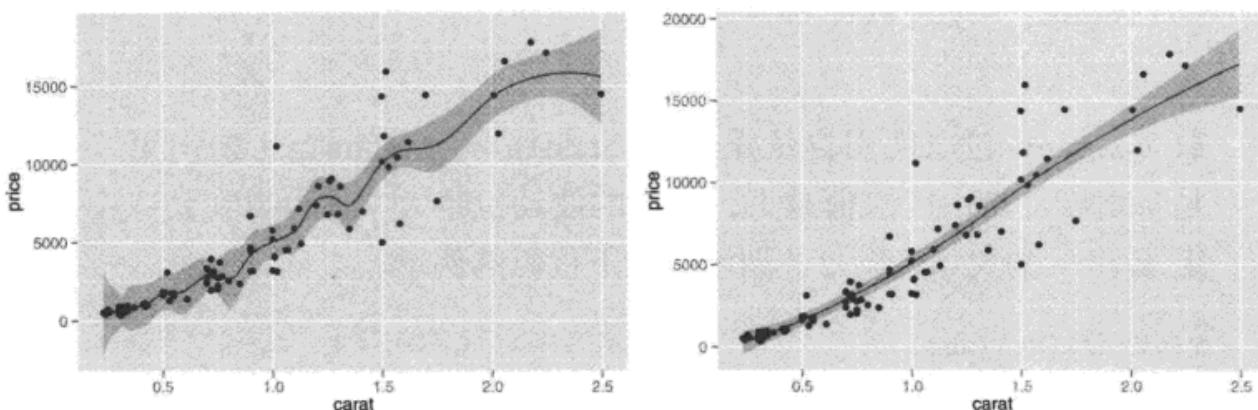


图 2.5 `span` 参数的作用。左图是 `span = 0.2`, 右图是 `span = 1`。

Loess 对于大数据并不十分适用 (内存的消耗是 $O(n^2)$), 因此当 n 超过 1000 时将默认采用另一种平滑算法。

- 你可以使用 `method = "gam"`, `formula = y ~ s(x)` 来调用 `mgcv` 包拟合一个广义可加模型。这与在 `lm` 中使用样条相类似, 但样条的阶数是通过数据估计得到的。对于大数据, 请使用公式 `y ~ s(x, bs = "cs")`, 这是数据量超过 1000 时默认使用的选项。

```

1 library(mgcv)
2
3 qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
4   method = "gam", formula = y ~ s(x))
5
6 qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
7   method = "gam", formula = y ~ s(x, bs = "cs"))

```

- `method = "lm"` 拟合的是线性模型。默认情况下会拟合一条直线, 但你可以通过指定 `formula = y ~ poly(x, 2)` 来拟合一个二次多项式或加

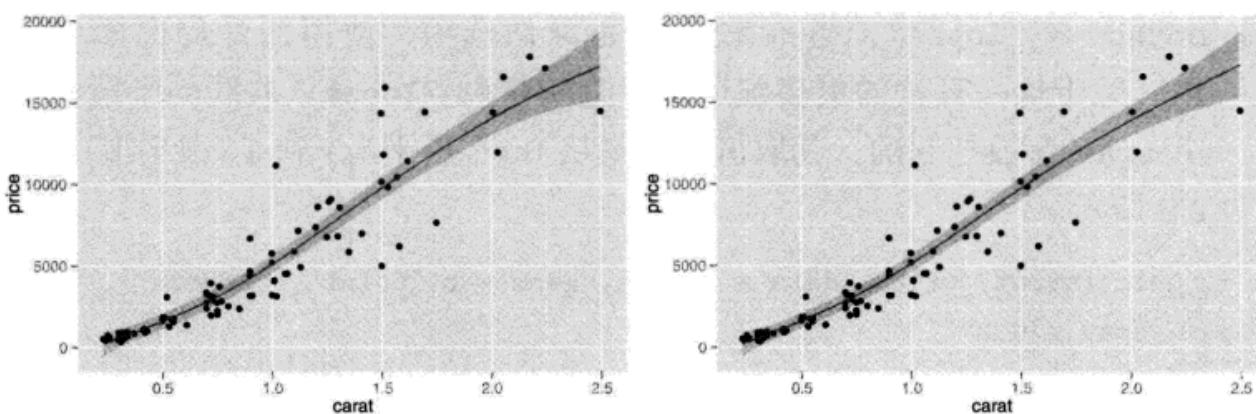


图 2.6 在运用广义可加模型作为平滑器时 formula 参数的作用。左图是 `formula = y ~ s(x)`，右图是 `formula = y ~ s(x, bs = "cs")`。

载 `splines` 包以使用自然样条：`formula = y ~ ns(x, 2)`。第二个参数是自由度：自由度取值越大，曲线的波动也越大。你可以在公式中任意指定 x 和 y 的关系，图 2.7 展示了如下代码的例子。

```

1 library(splines)
2
3 qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
4       method = "lm")
5
6 qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
7       method = "lm", formula = y ~ ns(x, 5))

```

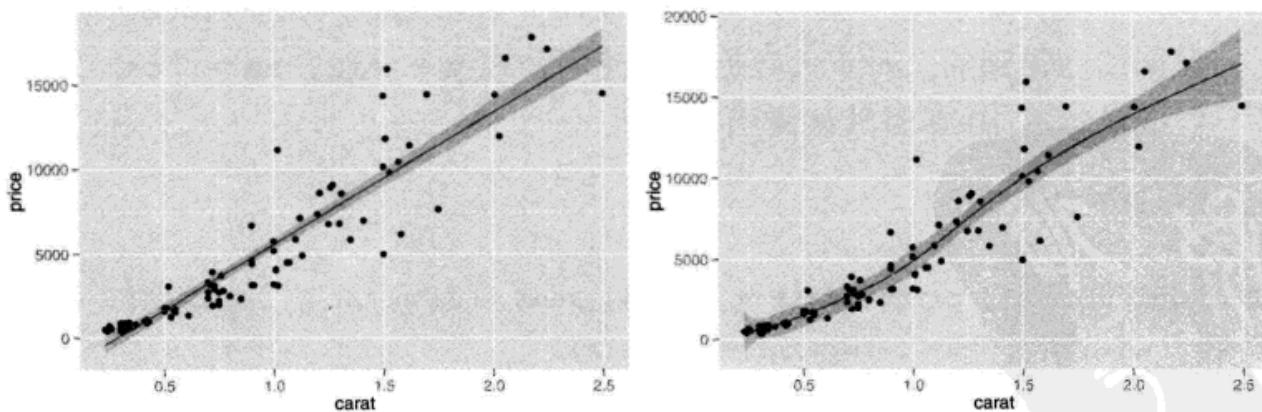


图 2.7 在运用线性模型作为平滑器时 formula 参数的作用。左图是 `formula = y ~ x` 的默认值，右图是 `formula = y ~ ns(x, 5)`。

- `method = "rlm"` 与 `lm` 类似，但采用了一种更稳健的拟合算法，使得结

果对异常值不太敏感。这一方法是 MASS 包的一部分，因此需要先加载 MASS 包。

2.5.2 箱线图和扰动点图

如果一个数据集中包含了一个分类变量和一个或多个连续变量，那么你可能会想知道连续变量会如何随着分类变量水平的变化而变化。箱线图和扰动点图提供了各自的方法来达到这个目的，图 2.8 展示了钻石每克拉的价格随颜色的变化情况，左图的扰动点图使用的是 `geom = "jitter"`，右图的箱线胡须图使用的是 `geom = "boxplot"`。

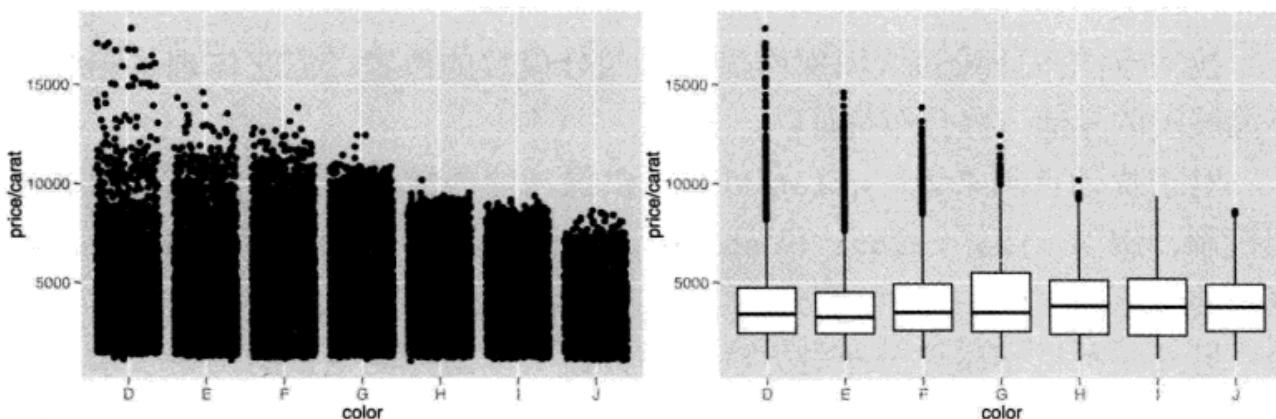


图 2.8 利用扰动点图（左）和箱线图（右）来考察以颜色为条件的每克拉价格的分布。随着颜色的改变（从左到右），每克拉价格的跨度逐渐减小，但分布的中位数没有明显的变化。

每一种方法都有它的优势和不足。箱线图只用了 5 个数字对分布进行概括，而扰动点图可以将所有的点都绘制到图中（当然会有图形重叠的问题）。在本例中，两种图形都显示出每克拉价格的跨度与钻石颜色是相关的，但箱线图的信息更充分，它显示出分布的中位数和四分位数都没有太大的变化。

扰动点图中的图形重叠问题可以通过半透明颜色来部分解决，也就是使用 `alpha` 参数。图 2.9 展示了三种水平的透明度，这使得我们可以更清楚地看出数据集中的地方。这三张图是用如下的代码生成的：

```

1 qplot(color, price / carat, data = diamonds, geom = "jitter",
2   alpha = I(1 / 5))
3 qplot(color, price / carat, data = diamonds, geom = "jitter",
4   alpha = I(1 / 50))
5 qplot(color, price / carat, data = diamonds, geom = "jitter",
6   alpha = I(1 / 200))

```

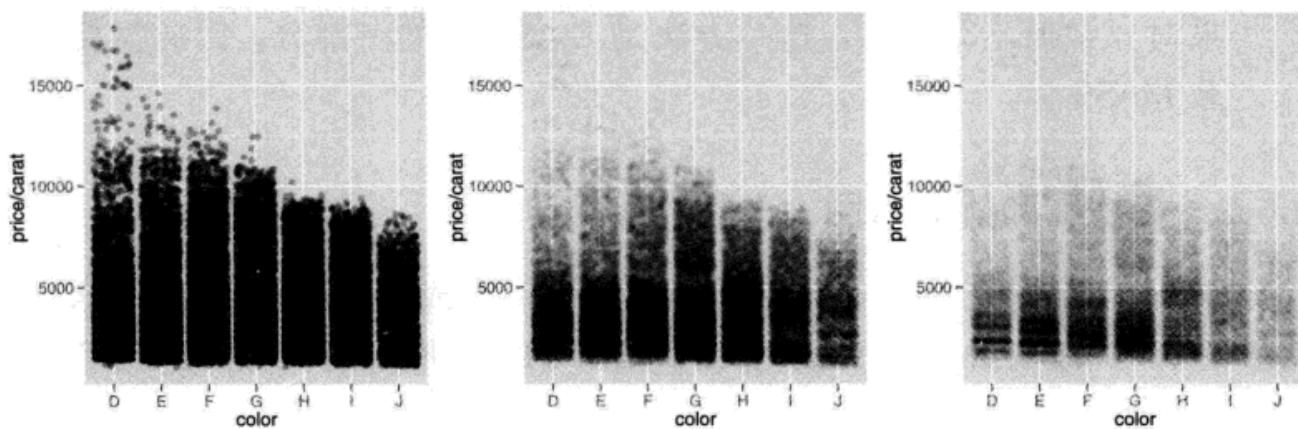


图 2.9 改变 alpha 的取值，从左到右分别为 $1/5$, $1/50$ 和 $1/200$ 。随着不透明度的降低，我们可以看出数据集中的地方。然而，箱线图依然是一个更好的选择。

扰动点图不能像箱线图那样显示出四分位数的位置，但它体现了箱线图所不能展示的其他一些分布特征。

对于扰动点图来说，`qplot` 可以提供像一般的散点图那样对其他图形属性的控制，例如 `size`, `colour` 和 `shape`。对于箱线图，你可以用 `colour` 控制外框线的颜色，用 `fill` 设置填充颜色，以及用 `size` 调节线的粗细。

另外一种考察条件分布的方法是用分面来作出分类变量的每一个水平下连续变量的直方图或密度曲线，这在 2.6 节中会进行介绍。

2.5.3 直方图和密度曲线图

直方图和密度曲线图可以展示单个变量的分布，相对于箱线图而言，它们提供了更多的关于单个分布的信息，但它们不太容易在不同组之间进行比较（尽管我们将会看到一种可行的办法）。图 2.10 展示了钻石重量的直方图和密度曲线图。

```
1 qplot(carat, data = diamonds, geom = "histogram")
2 qplot(carat, data = diamonds, geom = "density")
```

对于密度曲线图而言，`adjust` 参数控制了曲线的平滑程度（`adjust` 取值越大，曲线越平滑）。对于直方图，`binwidth` 参数通过设定组距来调节平滑度。（切分位置同样可以通过 `breaks` 参数进行显式的指定。）绘制直方图或密度曲线时，对平滑程度进行试验非常重要。在直方图中，你应该尝试多种组距：当组距较大时，图形能反映数据的总体特征；当组距较小时，则能显示出更多的细节。

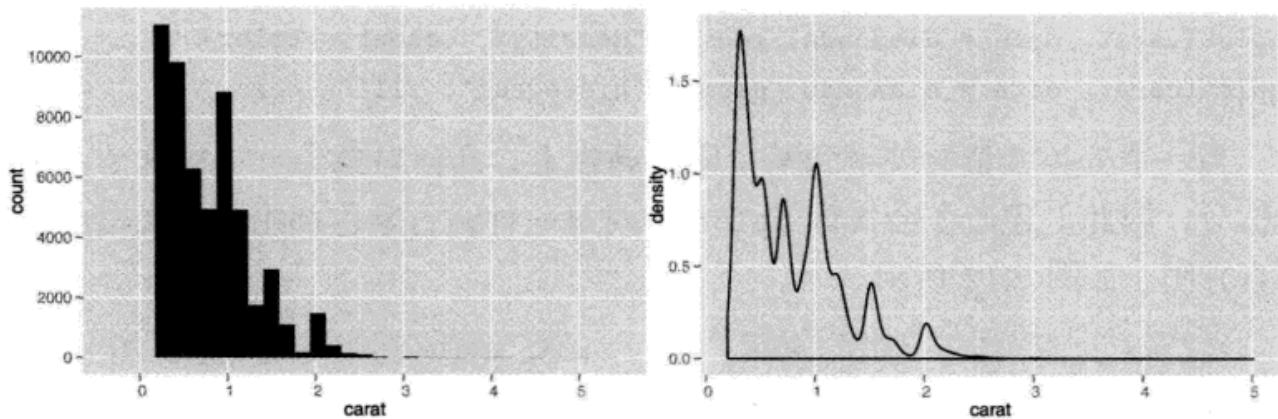


图 2.10 展示钻石重量的分布。左图使用的是 `geom = "histogram"` 右图使用的是 `geom= "density"`。

在图 2.11 中，我们尝试了三种 `binwidth` 的取值：1.0，0.1 和 0.01。只有在最小组距的图中（右），我们能发现之前在散点图中出现的竖直方向的条纹，它们主要集中在“整 0.5”的克拉数附近。完整的代码如下所示：

```

1 qplot(carat, data = diamonds, geom = "histogram", binwidth = 1,
2   xlim = c(0,3))
3 qplot(carat, data = diamonds, geom = "histogram", binwidth = 0.1,
4   xlim = c(0,3))
5 qplot(carat, data = diamonds, geom = "histogram", binwidth = 0.01,
6   xlim = c(0,3))

```

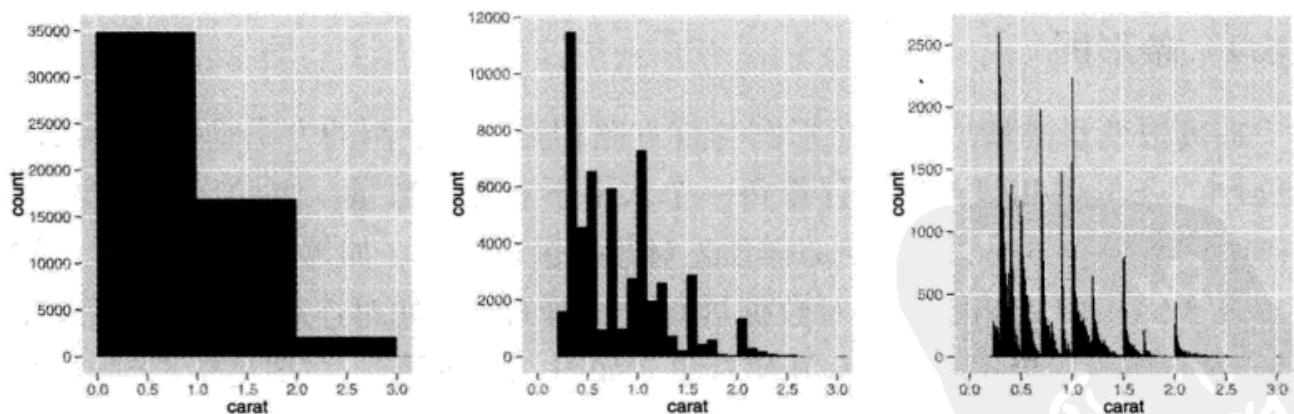


图 2.11 变动直方图的组距可以显示出有意思模式。从左到右，组距分别为 1, 0.1 和 0.01。只有重量在 0 到 3 克拉之间的钻石显示在图中。

要在不同组之间对分布进行对比，只需要再加上一个图形映射，如下面的代码所示。

```

1 qplot(carat, data = diamonds, geom = "density", colour = color)
2 qplot(carat, data = diamonds, geom = "histogram", fill = color)

```

当一个分类变量被映射到某个图形属性上，几何对象会自动按这个变量进行拆分，因此上述命令会告诉 `qplot()` 对每一种钻石颜色都绘制一次密度曲线和直方图，如图 2.12 所示。

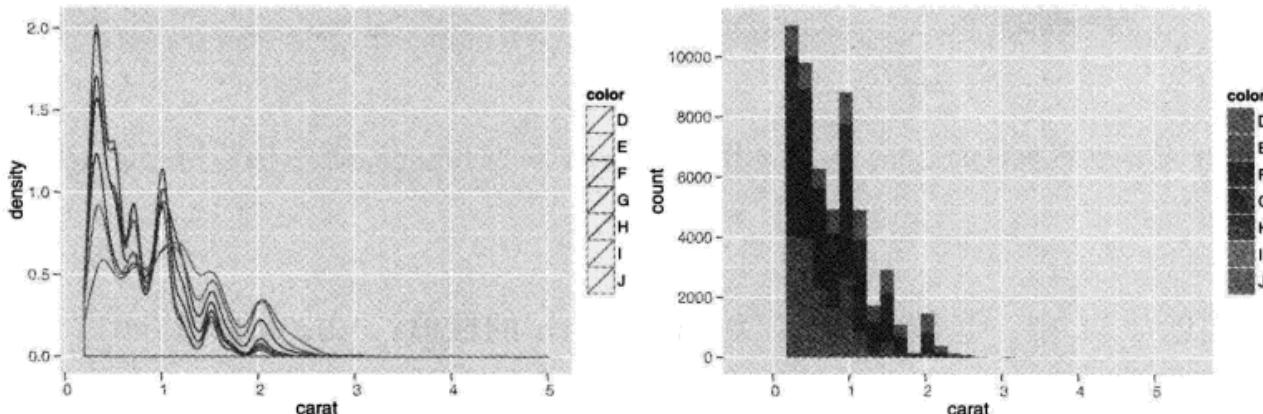


图 2.12 (参见彩图 3) 当一个分类变量被映射到某个图形属性上，几何对象会自动按这个变量进行拆分。左图是重叠的密度曲线图，右图是堆叠起来的直方图。

密度曲线图在第一眼看来更吸引人，因为它似乎很容易阅读，而且适于在不同的曲线之间进行比较。然而，要真正理解密度曲线的含义则比较困难，而且密度曲线有一些隐含的假设，例如曲线应该是无界、连续和平滑的，而这些假设不一定适用于真实的数据。

2.5.4 条形图

在离散变量的情形下，条形图与直方图相类似，绘制的方法是使用 `geom = "bar"`。条形图几何对象会计算每一个水平下观测的数量，因此你不需要像在基础绘图系统的 `barchart` 中那样预先对数据进行汇总。如果数据已经进行了汇总，或者你想用其他的方式对数据进行分组处理（例如对连续变量进行分组求和），那么你可以使用 `weight` 几何对象，如图 2.13 所示。左图是钻石颜色的普通条形图，右图是按重量加权的条形图。

```

1 qplot(color, data = diamonds, geom = "bar")
2
3 qplot(color, data = diamonds, geom = "bar", weight = carat) +
4   scale_y_continuous("carat")

```

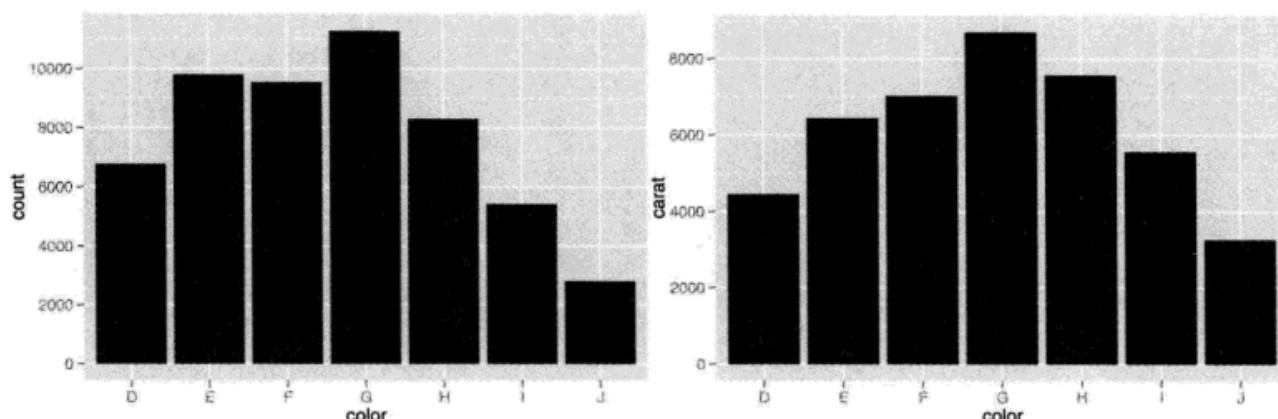


图 2.13 钻石颜色的条形图。左图显示的是分组的计数，右图是按 `weight = carat` 进行加权，展示了每种颜色的钻石的总重量。

2.5.5 时间序列中的线条图和路径图

线条图和路径图常用于可视化时间序列数据。线条图将点从左到右进行连接，而路径图则按照点在数据集中的顺序对其进行连接（线条图就等价于将数据按照 x 取值进行排序，然后绘制路径图）。线条图的 x 轴一般是时间，它展示了单个变量随时间变化的情况。路径图则展示了两个变量随时间联动的情况，时间反映在点的顺序上。

由于钻石数据中没有包含时间变量，因此在这里我们使用 `economics` 数据集，它包含了美国过去 40 年的经济数据。图 2.14 展示了失业水平随时间变化的两张线条图，它们是用 `geom = "line"` 进行绘制的。第一张图显示了失业率的变化，第二张图是失业星期数的中位数。我们已经可以看出这两个变量之间的一些区别，例如在最后的一个峰值处，失业的比例要比前一个峰值低，但失业的时间却要更长。

```
1 qplot(date, unemploy / pop, data = economics, geom = "line")
2 qplot(date, uempmed, data = economics, geom = "line")
```

要考察这种关系的更多细节，我们可以将两个时间序列绘制在同一张图中。尽管我们可以用一张散点图来表示失业率和失业时间长度之间的关系，但我们并不能从中看出变量随时间的变化。对此，解决的办法是将临近时点的散点连接起来，形成一张路径图。

在下面我们将画出失业率和失业时间长度随时间变化的路径。由于线条有

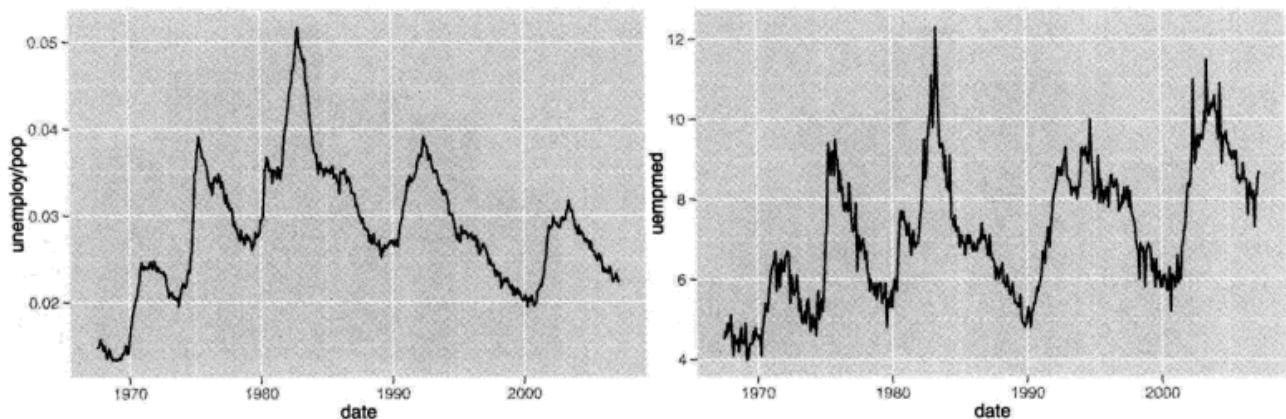


图 2.14 衡量失业程度的两张时序图。左图是失业人口的比例，右图是失业星期数的中位数。图形是用 `geom = "line"` 进行绘制的。

很多交叉，因此在第一张图中时间变化的方向并不明显。在第二张图中，我们将年份映射到了 `colour` 属性上，这让我们能更容易地看出时间的行进方向。

```

1 year <- function(x) as.POSIXlt(x)$year + 1900
2 qplot(unemploy / pop, uempmed, data = economics,
3       geom = c("point", "path"))
4 qplot(unemploy / pop, uempmed, data = economics,
5       geom = "path", colour = year(date))

```

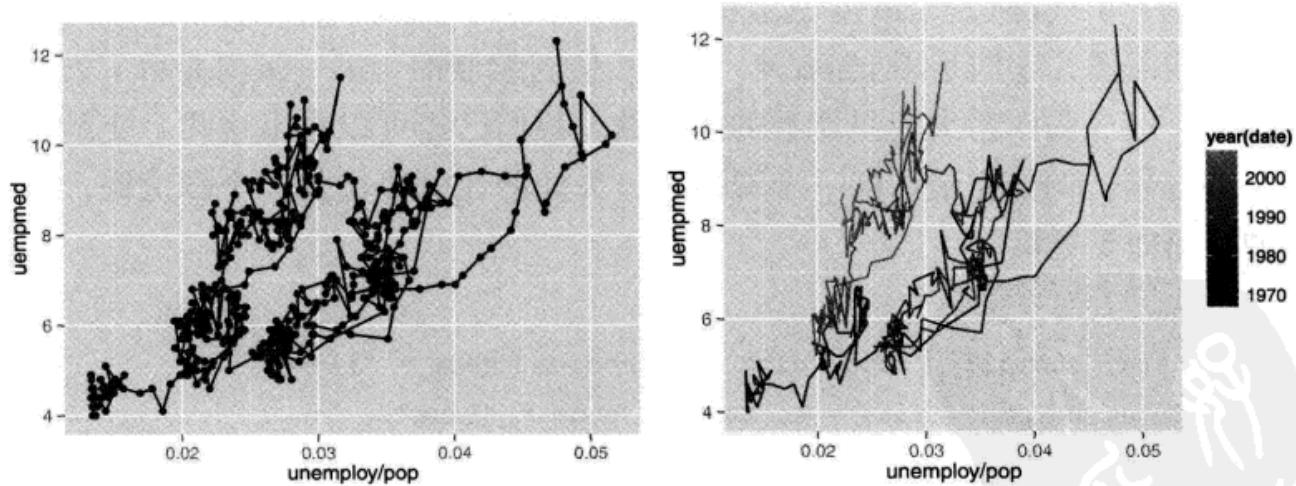


图 2.15 (参见彩图 2) 展示失业率和失业时间长度之间关系的路径图。左图是重叠在一起的的散点图和路径图，右图只有路径图，其中年份用颜色进行了展示。

我们可以看出来失业率和失业时间长度是高度相关的，尽管最近几年，失业时间长度与失业率相比有增长的趋势。

对于纵向数据，你可能想把多个时间序列画在同一张图中，每一个序列代表一个个体。要用 `qplot()` 实现这一想法，你需要将 `group` 图形属性映射到一个表示分组的变量之上。这一部分内容在 4.5.3 节有更深入的介绍。

2.6 分面

我们已经讨论了利用 **图形属性** (颜色和形状) 来比较不同分组的方法，它可以将所有的组绘制在同一张图中。**分面**是另外一种实现的方法：它将数据分割成若干子集，然后创建一个图形的矩阵，将每一个子集绘制到图形矩阵的窗格中。所有子图采用相同的图形类型，并进行了一定的设计，使得它们之间方便进行比较。7.2 节详细讨论了分面，包括它相对于图形属性分组的优势和劣势 (7.2.5 节)。

`qplot()` 中默认的分面方法是将图形拆分成若干个窗格，这可以通过形如 `row_var ~ col_var` 的表达式进行指定。你可以指定任意数量的行变量和列变量，但请注意当变量数超过两个时，生成的图形可能会非常大，以至于不适合在屏幕上显示。如果只想指定一行或一列，可以使用 `.` 作为占位符，例如 `row_var ~ .` 会创建一个单列多行的图形矩阵。

图 2.16 用了两张图来展示这个技巧，它们是以颜色为条件的重量的直方图。第二列的直方图绘制的是比例，这使得比较不同组的分布时不会受该组样本量大小的影响。左边一列直方图的 `y` 轴并不是原始数据的取值，而是将数据进行分组后的计数；`..density..` 则是一个新的语法，它告诉 `ggplot2` 将密度而不是频数映射到 `y` 轴。

```
1 qplot(carat, data = diamonds, facets = color ~ .,
2       geom = "histogram", binwidth = 0.1, xlim = c(0, 3))
3
4 qplot(carat, ..density.., data = diamonds, facets = color ~ .,
5       geom = "histogram", binwidth = 0.1, xlim = c(0, 3))
```

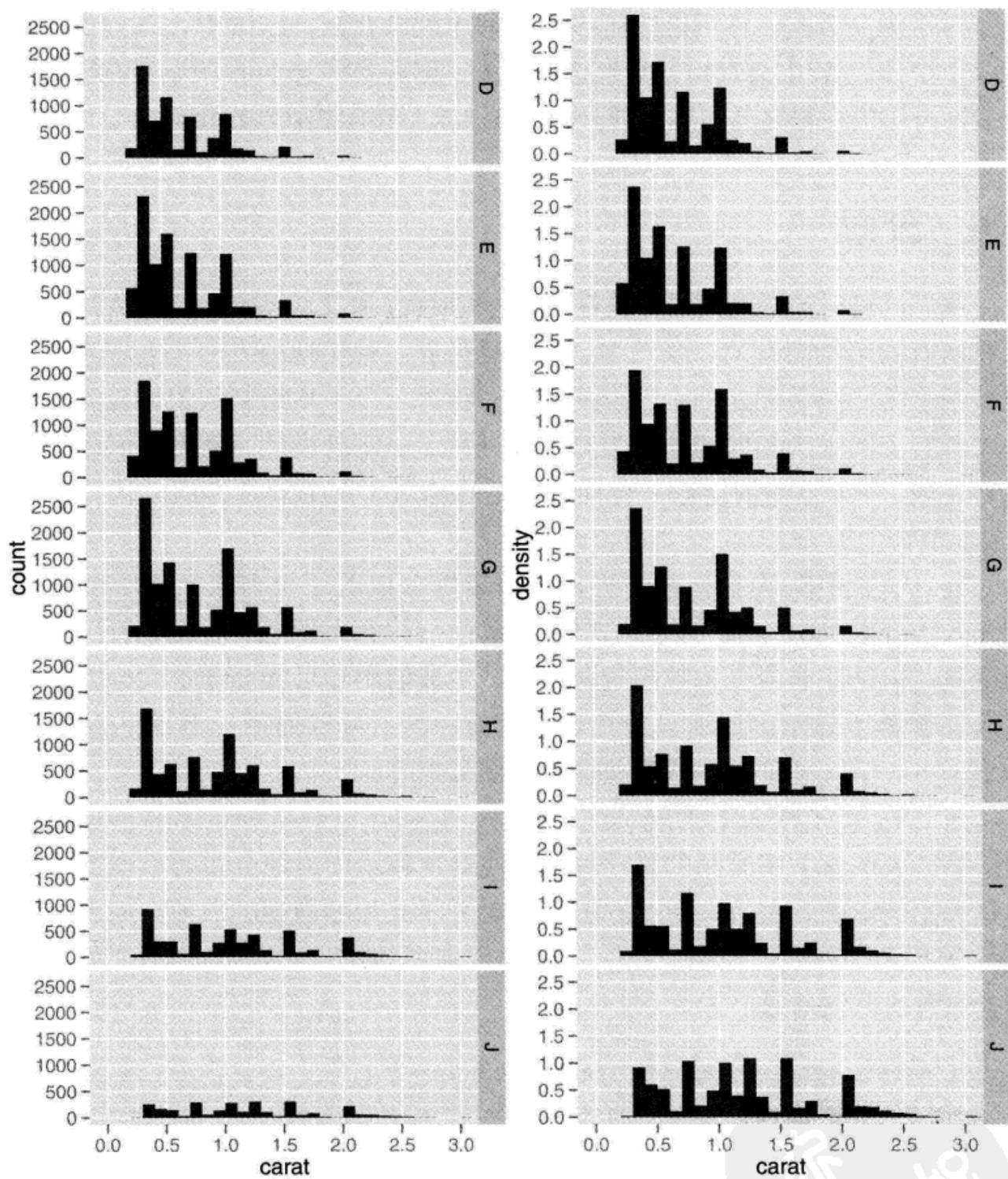


图 2.16 展示以颜色为条件的重量的直方图。左图展示的是频数，右图展示的是频率。频率图可以使得比较不同组的分布时不会受该组样本量大小的影响。高质量的钻石(颜色 D)在小尺寸上的分布是偏斜的，而随着质量的下降，重量的分布会变得越来越平坦。

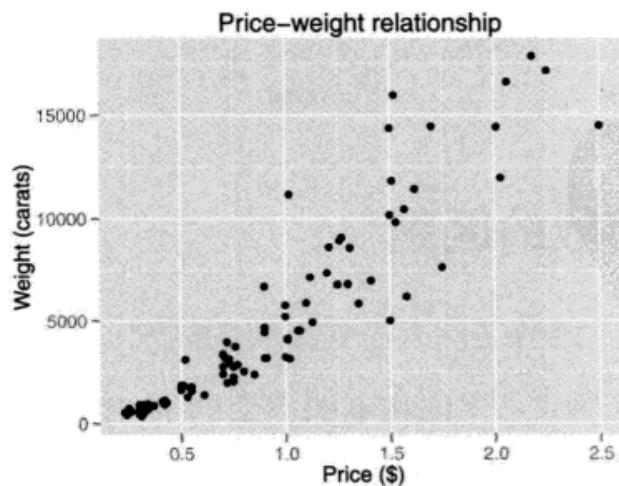
2.7 其他选项

`qplot` 中还有一些其他的选项用于控制图形的外观。这些参数与它们在 `plot` 中的作用相同：

- `xlim, ylim`: 设置 x 轴和 y 轴的显示区间，它们的取值都是一个长度为 2 的数值向量，例如 `xlim = c(0, 20)` 或 `ylim = c(-0.9, -0.5)`；
- `log`: 一个字符型向量，说明哪一个坐标轴（如果有的话）应该取对数。例如，`log = "x"` 表示对 x 轴取对数，`log = "xy"` 表示对 x 轴和 y 轴都取对数；
- `main`: 图形的主标题，放置在图形的顶端中部，以大字号显示。该参数可以是一个字符串（例如，`main = "plot title"`）或一个表达式（例如 `main = expression(beta[1] == 1)`）。可以运行 `?plotmath` 命令来查看更多数学表达式的例子；
- `xlab, ylab`: 设置 x 轴和 y 轴的标签文字，与主标题一样，这两个参数的取值可以是字符串或数学表达式。

下面的例子展示了这些选项的实际操作。

```
1 qplot(  
2   carat, price, data = dsmall,  
3   xlab = "Price ($)", ylab = "Weight (carats)",  
4   main = "Price-weight relationship"  
5 )
```

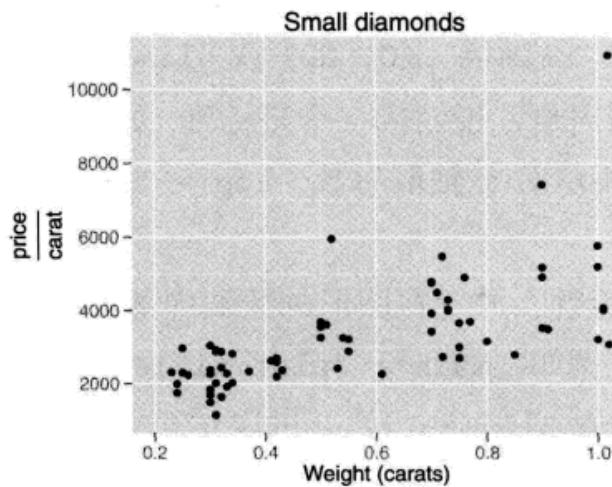


```
1 qplot(  
2   carat, price/carat, data = dsmall,
```

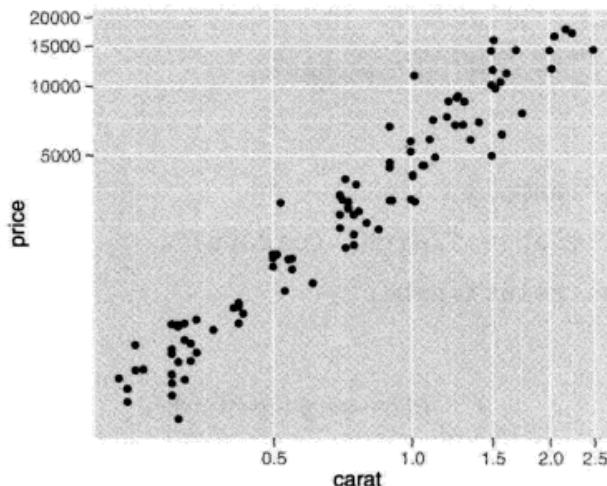
```

3   ylab = expression(frac(price, carat)),
4   xlab = "Weight (carats)",
5   main = "Small diamonds",
6   xlim = c(.2, 1)
7 )

```



```
1 qplot(carat, price, data = dsmall, log = "xy")
```



2.8 与 plot 函数的区别

plot 函数和 qplot 函数之间有一些重要的区别：

- qplot 不是泛型函数^①：当你将不同类型的 R 对象传入 qplot 时，它并不

^①译者注：0.9.0 版本之后引入了新的泛型函数 `autoplot`，此函数用来根据特定数据结构生成特定的完整图形。其默认定义是报错，该函数允许开发人员在其他包中对其针对不同对象进行不同定义，从而可以实现面向对象的图形设计。其功能类似于 `qplot`，可以同样封装所需的参数，区别是当你将不同类型的 R 对象传入 `autoplot` 时，它可以自动匹配默认的函数调用。

会匹配默认的函数调用。需要注意的是，`ggplot()` 是一个泛型函数，以它为起点，你可以对任意类型的 R 对象进行可视化操作。第 9 章对此有详细的描述；

- 一般而言，你可以将一个变量传递给你感兴趣的图形属性，这样该变量将进行标度转换并显示在图例上。如果你想对其进行赋值，比如让点的颜色变为红色，则可以使用 `I()` 函数：`colour = I("red")`。这部分内容在 4.5.2 节中有进一步的介绍；
- `ggplot2` 中的图形属性名称（如 `colour`, `shape` 和 `size`）比基础绘图系统中的名称（如 `col`, `pch` 和 `cex` 等）更直观，且更容易记忆；
- 在基础绘图系统中，你可以通过 `points()`, `lines()` 和 `text()` 函数来向已有的图形中添加更多的元素。而在 `ggplot2` 中，你需要在当前的图形中加入额外的图层，这将在下一章中进行介绍。

第 3 章 语法突破

3.1 简介

如果不想深入研究底层的语法，你可以选择只使用 `qplot()`，但是这样你将永远无法掌握 `ggplot2` 的精髓。通过深入学习语法及其组成部件，你将不仅能绘制出更多种类的图形，而且还能整合多个数据源，自定义出理想的图形。初读此书时你可以先略过此章，等想更深入理解图形各部分是如何恰当整合在一起时，再回头研读。

本章主要介绍 `ggplot2` 的理论基础：图形图层语法。图形图层语法基于 Wilkinson 的图形语法 (Wilkinson, 2005)，并在其基础上添加了许多新功能，使得图形更有表现力，并能完美地嵌入到 R 环境中。有关图层语法和 Wilkinson 图形语法的不同之处在 Wickham(2008) 有详细描述，GPL(SPSS 的图形编码) 和 `ggplot2` 间的转换指南可参见附录 A。在本章中，你将简单了解一些语法组成组件的基本知识，学习如何将它们整合到一起。下一章我们将讨论更多语法组件的细节，并给出更多实际应用的示例。

无论是对于普通用户还是对于统计图形开发人员，图层语法都是非常有用的。对于普通用户而言，它使得图形的重复更新变得更简单——每次只更新一个特征。该语法的价值还在于它从更高的视角审视了图形的构成，它认为图形的每个组件都是可以被修改的。因此，这就给了我们一个作图的基本结构框架，使得图形的绘制更为容易。对于特殊的问题，我们还可以利用它很方便地定义新图形。

对于开发人员而言，图层语法使得向 `ggplot2` 里添加新功能变得更方便。你只需要添加你所需要的新图形组件，而所有其他现存的组件可以继续使用。例如，你可以添加一个新的统计变换，并且继续使用原有的标度和几何对象。另

外，图层语法也有助于发现新类型的图形，因为它有效地定义了统计图形的参数空间。

本章首先详述了绘制简单图形的过程。然后，3.3 节介绍了简单散点图的绘制方法，3.4 节在此基础上又添加了光滑曲线和分面。在学习这些例子的同时，六种语法组件也会被先后提及。随后 3.5 节对它们作了更为精确的定义。最后，本章以 3.6 节作结，介绍了怎样将这些图形组件映射成 R 的数据结构。

3.2 耗油量数据

表 3.1 列出了耗油量数据集 `mpg` 的一个样本。该数据集记录了美国 1999 年和 2008 年部分汽车的制造厂商、型号、类别、引擎大小、传动系和耗油量等信息。该数据集包含 38 种型号的汽车，包括非常流行的 Audi A4, Honda Civic, Hyundai Sonata, Nissan Maxima, Toyota Camry 和 Volkswagen Jetta。该数据来源于 EPA 燃油经济性网站，<http://fueleconomy.gov>。

表 3.1 `ggplot2` 包里 `mpg` 数据集的前 10 行数据，`cty` 和 `hwy` 分别记录了城市和高速公路驾驶时的耗油量（英里每加仑，`mpg`），`displ` 表示发动机排量（以升为单位）。

manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
audi	a4	1.80	1999	4	auto(l5)	f	18	29	p	compact
audi	a4	1.80	1999	4	manual(m5)	f	21	29	p	compact
audi	a4	2.00	2008	4	manual(m6)	f	20	31	p	compact
audi	a4	2.00	2008	4	auto(av)	f	21	30	p	compact
audi	a4	2.80	1999	6	auto(l5)	f	16	26	p	compact
audi	a4	2.80	1999	6	manual(m5)	f	18	26	p	compact
audi	a4	3.10	2008	6	auto(av)	f	18	27	p	compact
audi	a4 quattro	1.80	1999	4	manual(m5)	4	18	26	p	compact
audi	a4 quattro	1.80	1999	4	auto(l5)	4	16	25	p	compact
audi	a4 quattro	2.00	2008	4	manual(m6)	4	20	28	p	compact

该数据集隐含了许多有趣的问题。引擎大小和耗油量有什么关系？是不是某些制造商比其他的制造商更关注汽车的耗油量？耗油量在过去的十年中有没有明显的增加？我们将尝试回答第一个问题，并在此过程中学习更多关于绘制散点图的细节。

3.3 绘制散点图

观察图 3.1，它是对上面问题的一个简单回答。这是一个含有两个连续型变量的散点图（发动机排量（engine displacement）和高速公路每加仑行驶的英里数（highway mpg）），图中点的颜色是由第三个变量（汽缸（cylinders）的数目）决定的。利用以前章节里学的知识，你应该知道如何使用 `qplot()` 来作这幅图。但是这背后的具体原理是怎样的呢？`ggplot2` 是怎样作出这幅图的呢？

```
1 qplot(displ, hwy, data = mpg, colour = factor(cyl))
```

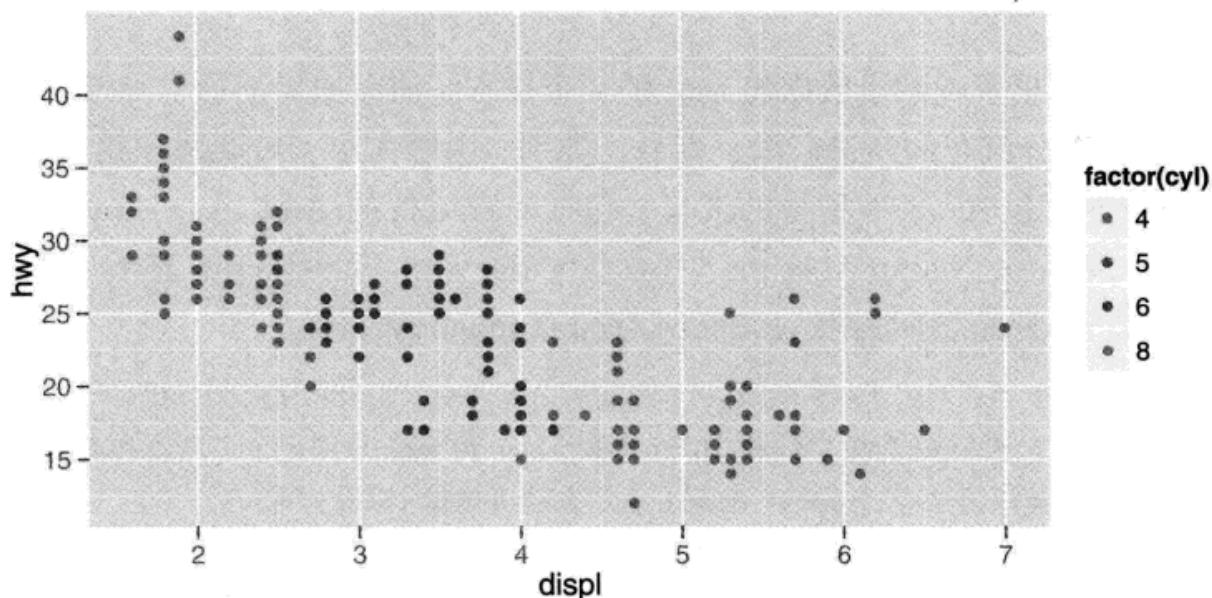


图 3.1 (参见彩图 4) 发动机排量（以升为单位，`displ`）对高速公路耗油量（英里每加仑，`hwy`）散点图。图中的点是根据汽缸的数目着色的。通过该图可以发现影响耗油量的一个重要因素：发动机的排量大小。

图形属性与数据的映射

我相信你已经见过许多的散点图，甚至还自己动手画过。但散点图的精确定义是什么呢？在散点图中，每一个观测数据都用一个点（.）来表示，点的位置由两个变量的值决定。每个点不仅有横坐标和纵坐标，还有大小、颜色和形状，这些属性我们称之为图形属性（aesthetics）。每个图形属性都可以映射为一个变量或者设定成一个常数。图 3.1 中变量 `displ` 控制点的水平位置，变量 `hwy` 控制点的竖直位置，变量 `cyl` 控制点的颜色，而点的大小和形状都没有指定映射的变量，使用的是默认值（常数）。

根据变量和图形属性的映射关系，我们可以新建一个记录这些信息的数据集。表 3.2 给出了图 3.1 所对应数据集的前 10 行数据。这个新数据集是在将图形属性映射到原数据集时产生的，利用该数据集我们可以作出许多不同种类的图。在散点图中，我们用点来表示每个观测，倘若用直线连接所有的观测，我们就会得到折线图。如果用条形来表示这些数据，我们便会得到条形图。对于该数据集，折线图和条形图是没有意义的，但是为了对比学习，我们也将其画出来，如图 3.2 所示。注意，在 ggplot2 中，我们可以作出很多没有意义但符合语法的图形，就好比在英语中，我们也可以写无语法错误但毫无意义的句子，比如“angry rock barked like a comma”（愤怒的石头像逗号般狂叫）。

表 3.2 数据集 mpg 的前十行数据，并被重新整理成散点图所需的格式。该数据框包含了绘制散点图所需的全部数据信息。

x	y	colour	x	y	colour
1.80	29	4	2.80	26	6
1.80	29	4	3.10	27	6
2.00	31	4	1.80	26	4
2.00	30	4	1.80	25	4
2.80	26	6	2.00	28	4

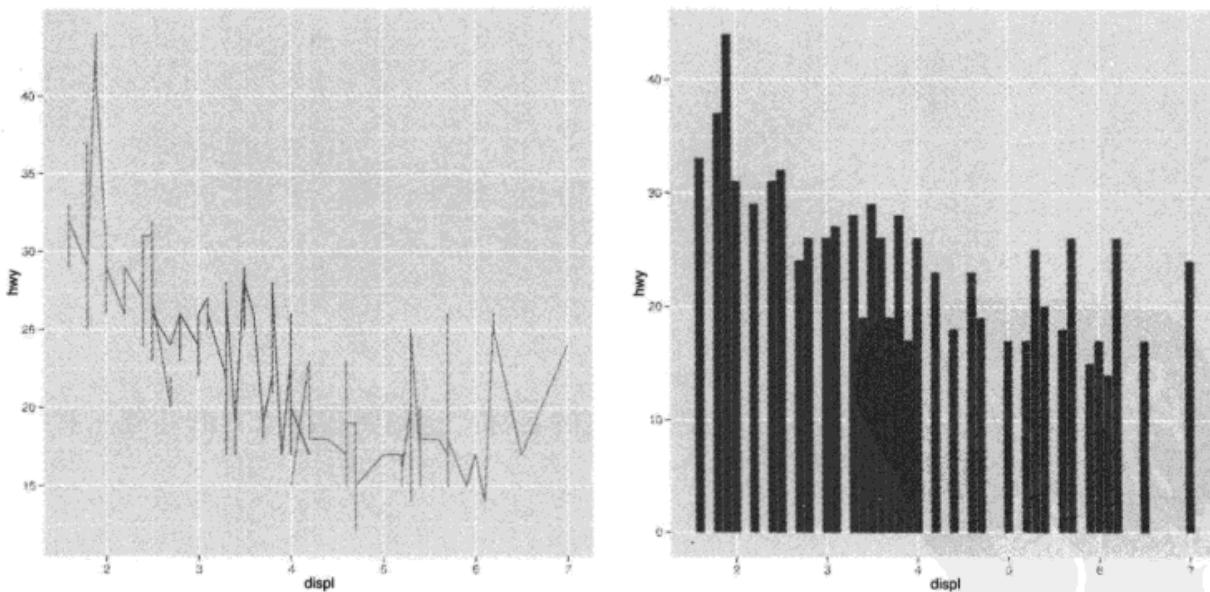


图 3.2 除了用点来表示数据，我们还可以用其他的的几何对象，如直线（左）或条形（右）。对于这个数据，用这两种几何对象所作的图形都没有什么意义，但在语法上是有效的。

点 (points)、线 (lines) 和条 (bars) 都是几何对象的具体形式，被称作 ge-

om。几何对象决定了图形的“类型”(type)。只含一种几何对象的图通常有特定的名字,见表3.3。由多种几何对象组合而成的更复杂的图形通常没有特定的名字,我们需要对其重新描述。例如,图3.3在原有的散点图上添加了分组回归线,那么这个图形该叫什么呢?一旦你掌握了ggplot2的语法学,你就会发现,根据具体问题所作的很多图形都没有特定的名字。

表3.3一部分有名称的图以及它们所对应的几何对象。

图形名称	几何对象(geom)	其他特征
散点图	点(point)	
气泡图	点	点的大小映射给其他变量
条形图	条形(bar)	
箱线图	箱子(boxplot)	
折线图	线(line)	

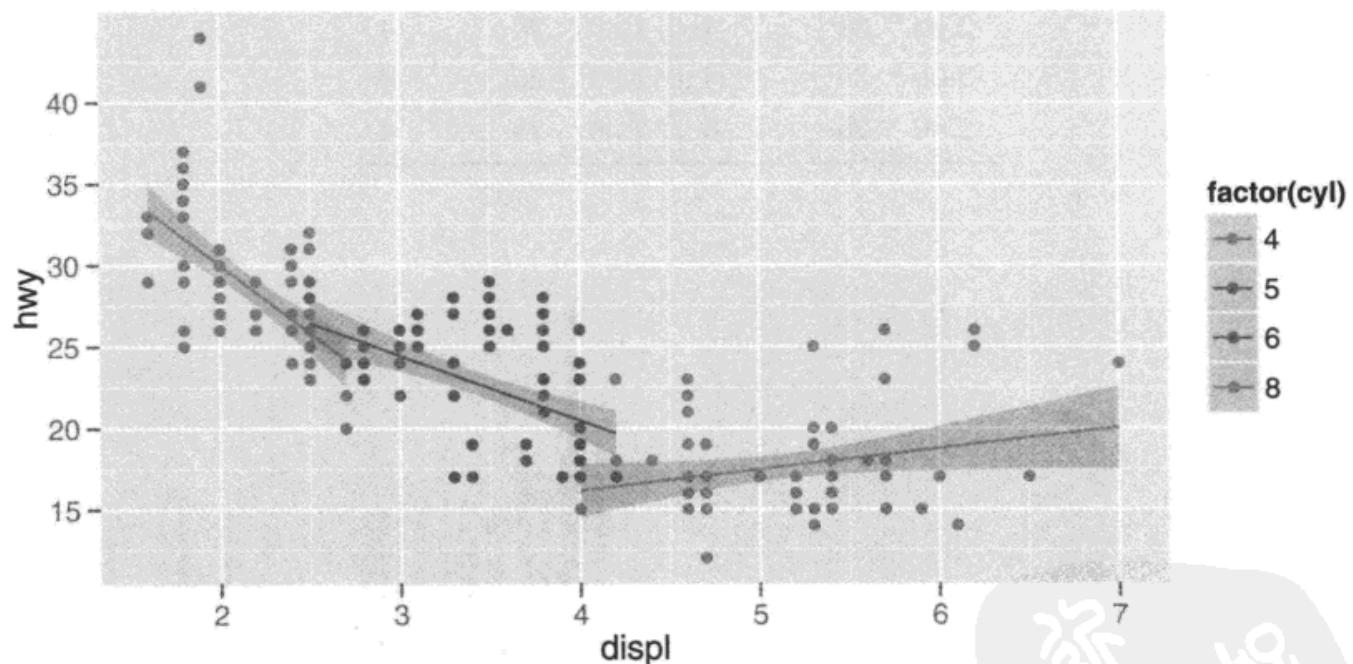


图3.3(参见彩图5)更复杂的图形一般没有特定的名字。这幅图在图3.1的基础上对每个组添加了回归线。这个图应该叫什么名字呢?

标度变换

表3.2中的数值对于电脑而言没有任何意义。我们需要把它们从数据单位(如升、英里每加仑和汽缸数)转换成电脑可以识别的物理单位(如像素和颜色)。

这个转换过程我们称之为标度变换 (scaling)。虽然转换后的数据对我们而言可能没有意义，但对电脑而言却是可识别的：颜色用 6 个字母组成的十六进制字符串来表示，大小和形状分别用数字和整数来表示。这些 R 可以识别的图形属性值可参见附录 B。

在这个例子中，我们有三个图形属性需要进行标度变换：水平位置 (x)、竖直位置 (y) 和颜色。在这个例子中，位置变换非常简单，因为我们用的是默认的线性变换，只需要将数据的范围线性映射到 $[0,1]$ 区间上即可。因为 `ggplot2` 调用的绘图系统 `grid` 包会帮我们处理好最终的转换细节，所以我们用 $[0,1]$ 而不是精确的像素值。最后一步是如何根据点的位置 (x 和 y) 来确定它在图中的位置，这是由坐标系统决定的，被称作 `coord`。大多数情况下我们将使用笛卡尔坐标系，但有时我们也会用极坐标系或用地图中的球投影。

颜色变换的过程就略微复杂些，因为我们将得到一个非数字的结果。颜色可以看作由三种组件组成，它们与人眼中识别颜色的三种细胞相对应。这三种细胞建立了一个三维的颜色空间，颜色的标度转换就是将数据的值映射到这个空间中。映射方法有很多，由于此处 `cyl` 是分类变量，我们便把它的值等距地映射到色轮上，如图 3.4 所示。当变量是连续型时，我们将使用其他映射方法。

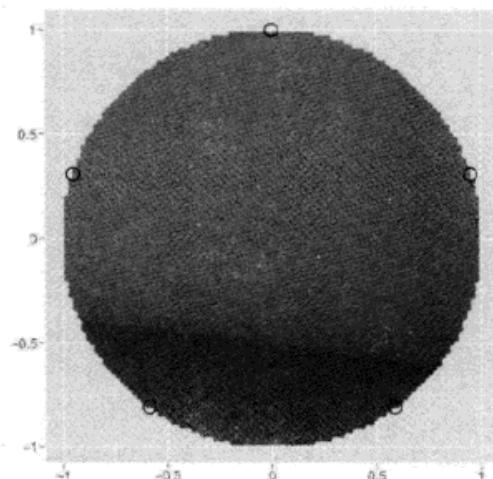


图 3.4 (参见彩图 6) 一个色轮。该图展示了如何获取五个等距的颜色。它是离散型变量默认颜色标度的取法。

标度变换后的结果见表 3.4，它所包含的值都是电脑能够识别的。表中不仅包含了映射到变量的图形属性，也包含了设定为常数的图形属性。这样每个点的图形属性都被完整设定，R 便可以绘出相应的图形。

最后，我们还需要对这些数据进行渲染，生成能在屏幕上展示的图形对象。

表 3.4 进行标度变换后得到的数据集。颜色图形属性是由十六进制字符表示的；形状图形属性使用默认值，R 中的形状 19；大小图形属性使用默认值，直径为 1 毫米。

x	y	颜色	大小	形状
0.037	0.531	#FF6C91	1	19
0.037	0.531	#FF6C91	1	19
0.074	0.594	#FF6C91	1	19
0.074	0.562	#FF6C91	1	19
0.222	0.438	#00C1A9	1	19
0.222	0.438	#00C1A9	1	19
0.278	0.469	#00C1A9	1	19
0.037	0.438	#FF6C91	1	19
0.037	0.406	#FF6C91	1	19
0.074	0.500	#FF6C91	1	19

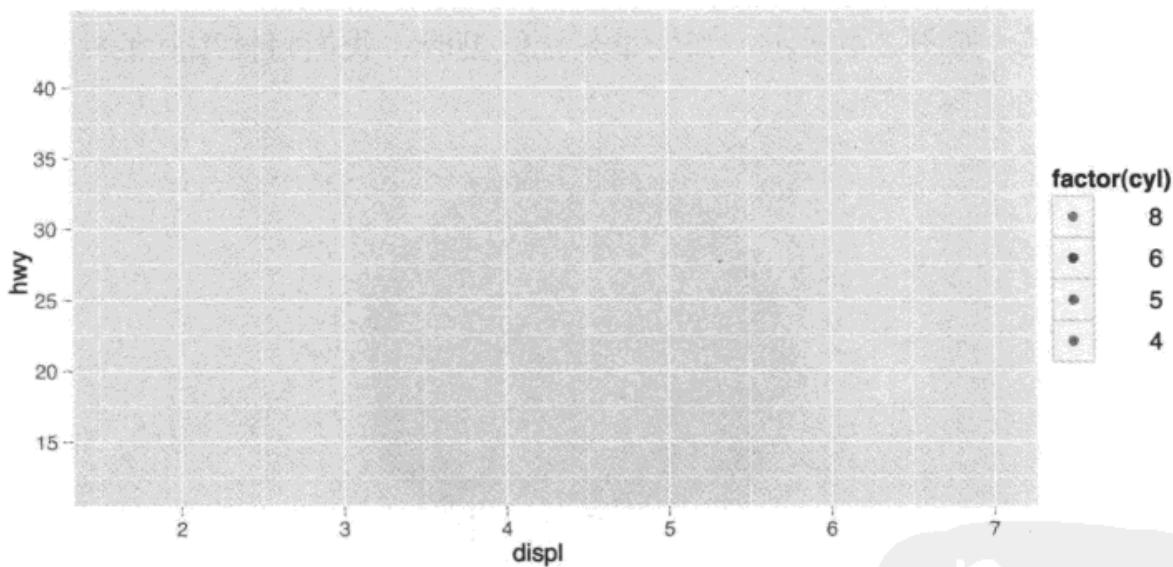


图 3.5 该图只保留了标度、坐标轴、图例、网格线和背景色。代表数据的点被移除。

绘制一个完整的图形，我们需要组合三类图形对象：数据，由点来表示；标度和坐标系，用来生成坐标轴和图例，通过它们我们才能读出图中蕴涵的信息；图形注释，如背景和标题。图 3.5 将数据从图形中分离出来，只留下了标度和图形注释。

3.4 更复杂的图形示例

我们先来看一个更复杂的图形(图 3.6)。这幅图添加了三种新的组件：分面、多个图层和统计量。与上一节所讲的类似，分面和图层将原数据切割成多个小数据集，即每个图层的每个分面面板(facet panel)都含一个小数据集。你可以把它想象成一个三维矩阵：分面面板形成了一个 2 维网格，图层在第三维的方向上叠加。本例中所有图层的数据都是一样的，但是一般而言，我们可以在不同的图层里使用不同的数据集。表 3.5 给出了每个分面里的前几行数据。

```
1 qplot(displ, hwy, data = mpg, facets = .~year) + geom_smooth()
```

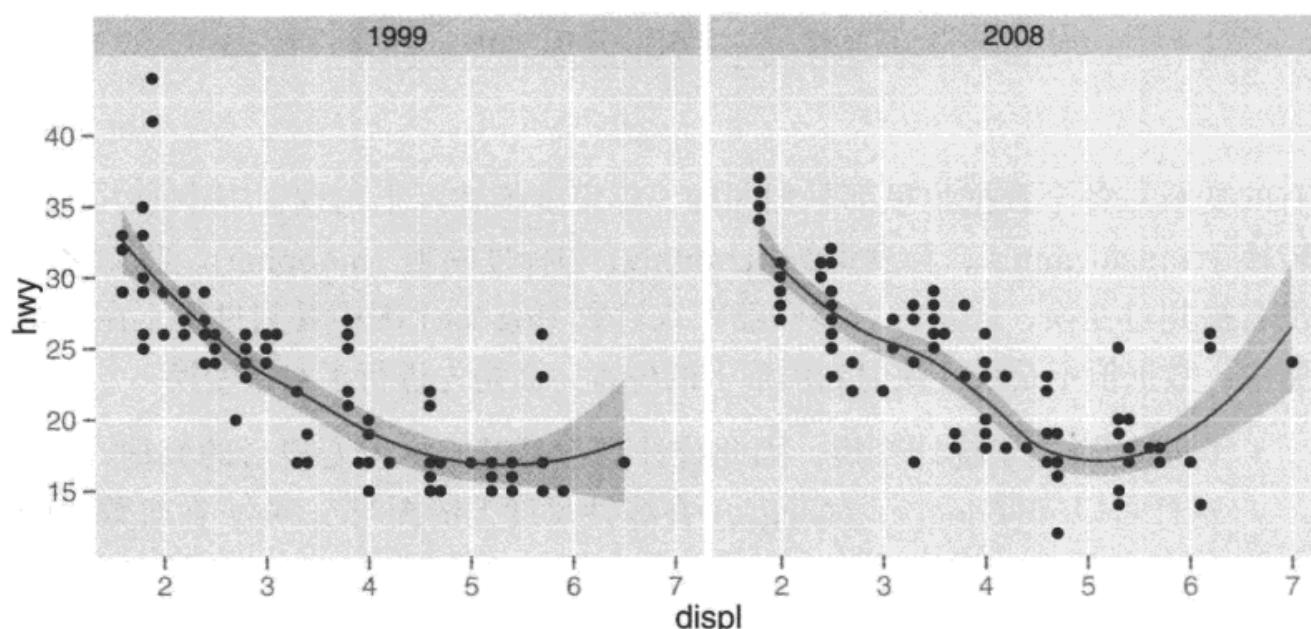


图 3.6 一个含有分面和多个图层的复杂图形。

平滑曲线层与散点层的不同点在于它没有展示原数据，而是展示了统计变换后的数据。特别地，平滑曲线层拟合了一条穿过数据中间位置的平滑曲线。添加该图层需要在我们前面介绍过的流程里再添加一步：将数据映射到图形属性后，要对其进行统计变换(对数据进行有效的处理)。在本例中，统计变换首先用一条 loess 平滑曲线来拟合数据，然后在数据的范围内，利用等间隔的点，计算并返回点所对应的预测值。其他有用的统计变换包括 1 维和 2 维的封箱(binning)，求组平均(group means)，分位数回归(quantile regression)和等高线(contouring)。

因为添加了统计变换，所以在进行标度变换时需要添加额外的步骤。这是因为现在我们有多个数据集(对不同的分面和图层而言)，所以我们需要确保所

表 3.5 分面使用的 1×2 数据框。一般该类型数据还有在第三维方向上叠加的图层数据，但在本例中每个图层的数据集都一样。

x	y	colour	x	y	colour
1.80	29	4	2.00	31	4
1.80	29	4	2.00	30	4
2.80	26	6	3.10	27	6
2.80	26	6	2.00	28	4
1.80	26	4	2.00	27	4
1.80	25	4	3.10	25	6
2.80	25	6	3.10	25	6
2.80	25	6	3.10	25	6
2.80	24	6	4.20	23	8
5.70	17	8	5.30	20	8

有的变换在各个数据集里都是相同的。标度变换实际上出现在三个地方：标度转换 (transforming)，标度训练 (training) 和标度映射 (mapping)。我们之前还没有提到过变换，但是你可能已经在双对数 (log-log) 图中看到过了。在双对数图中，数据值不是线性映射到图形上的位置，而是先进行了对数变换。

- 标度转换先于统计变换，因此统计量都是基于标度变换后的数据计算的。这样可以确保 $\log(x)$ 对 $\log(y)$ 在线性尺度上的图与 x 对 y 在对数尺度上的图看起来一样。另外，还有其他不同的变换可以使用，包括取平方根、对数和倒数。详见 6.4.2 节；
- 计算完统计量之后，所有分面和图层的数据集中的每个标度都会被“训练”。标度训练将根据所有小数据集里数据的范围得到整体数据的范围。如果没有这一步，标度将只具有局部意义，当将不同的层叠加到一起时，它们的位置就会错乱。不过有些时候我们也需要得到不同分面间（绝不是图层间）标度有错位的图形，这种情况详见 7.2.3 节；
- 最后，标度映射将数据映射到图形属性中。这是一个局部操作：每个数据集里的变量都映射给相应的图形属性值，生成一个新的数据集后再用几何对象来渲染。

图 3.7 显示了整个流程。

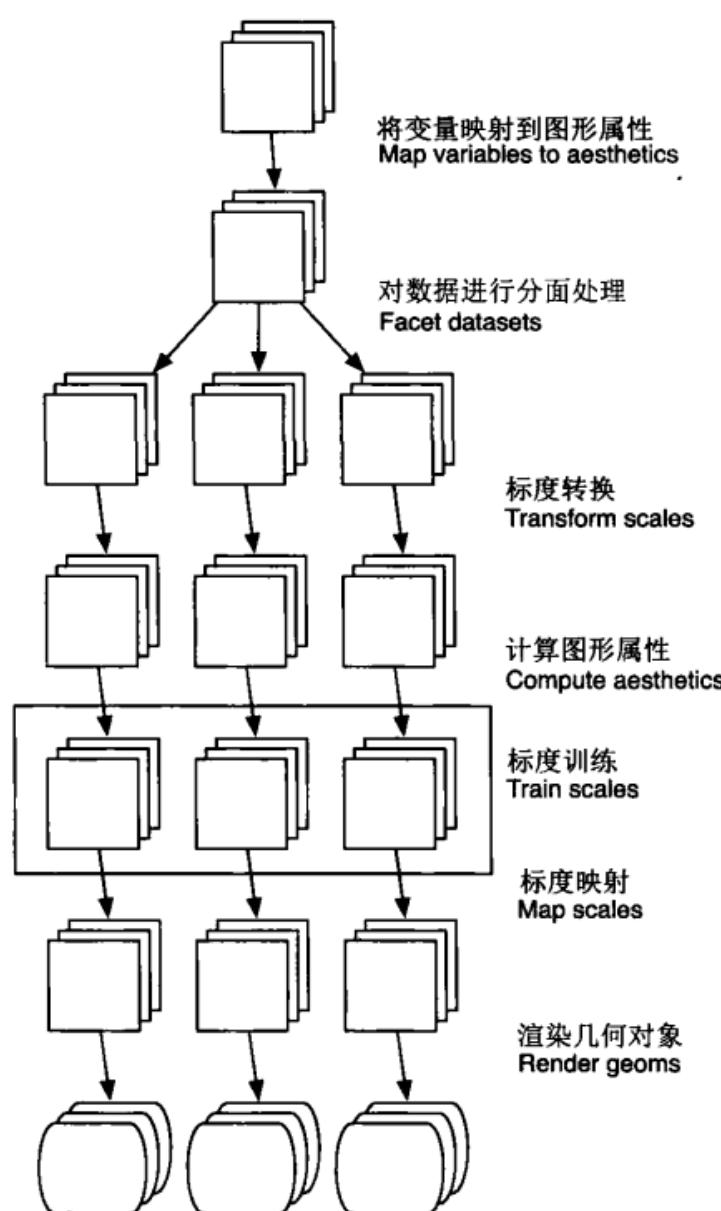


图 3.7 ggplot2 绘图过程图解。每个正方形代表一个图层，本图展示了一个含有三个图层和三个分面板的图。除了标度训练不对数据框做变换，其他步骤都对每个小数据集做变换。

3.5 图层语法的组件

在上面的例子中，我们已经见到了一些组成图像的组件，诸如数据、图形属性映射、几何对象、统计变换、标度和分面。我们也接触过坐标系。一个还没有提及的组件是位置调整，它控制着图形对象的重叠。数据、映射、统计变换、几何形状和位置调整共同组成了一个图层。一个图形可能含有很多图层，比如前面的例子中，我们把一个平滑曲线层添加到一个散点图层上。总的来说，图层语法所定义的图由以下几部分组成：

- 一个默认的数据集和一组从变量到图形属性的映射；
- 一个或多个图层，每个都由一种几何对象、一种统计变换和一种位置调整组成，另外数据集和图形属性映射也是可选的；
- 标度，每个图形属性映射都对应一个标度；
- 一个坐标系统；
- 分面设定。

下几节将对每种图层语法的组件作更详细的介绍，并给出书中各部分所参考的位置。

3.5.1 图层

图层的作用是生成在图像上可以被人感知的对象。一个图层由 4 部分组成：

- 数据和图形属性映射；
- 一种统计变换；
- 一种几何对象；
- 一种位置调整方式。

图层的属性将在第 4 章介绍，而第 5 章则介绍应用它们进行数据可视化的方法。

3.5.2 标度

标度控制数据到图形属性的映射，并且图形上所用的每一个图形属性都对应着一个标度。每个标度都作用于图形中的所有数据，以确保从数据到图形属性映射的一致性。一些标度如图 3.8 所示。

一个标度就是一个含有一组参数的函数，它的逆也是如此。例如颜色梯度标度，它把一条实线的各部分映射成一条含不同颜色的路径。函数中的参数可规定该路径是直线还是曲线，决定选择哪个颜色空间（例如，LUV 还是 RGB）、起始和终止位置的颜色。

其逆函数被用来绘制参照对象，通过参照对象你才能读出图里隐含的信息。参照对象可以是坐标轴（位置标度）或者是图例（其他标度）。大多数的映射都有唯一的逆函数（也就是一对一映射），但有些不是。逆映射的唯一性使得复原数据成为可能，但当我们只关注某个方面时，我们不会很在意它是不是一一映射。

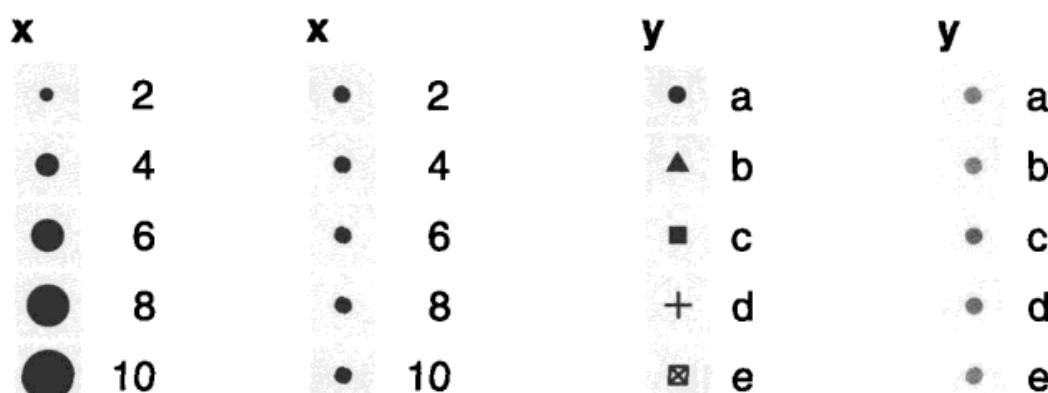


图 3.8 (参见彩图 7) 四种不同标度的图例。从左到右依次是：连续型变量映射到大小和颜色，离散型变量映射到形状和颜色。

第 6 章给出了标度更多的细节。

3.5.3 坐标系

坐标系，或简称为 `coord`，可将对象的位置映射到图形平面上。位置通常由两个坐标 (x, y) 决定，但是有时可能需要三个或更多（尽管目前还不能在 `ggplot2` 中实现）。笛卡尔坐标系是最常用的二维坐标系，极坐标系和各种地图投影则用得相对少一些。

坐标系可以同时影响所有的位置变量。与标度不同，坐标系还可以改变几何对象的外观。例如，在极坐标系中，条形看起来像扇形。另外，标度变换是在统计变换前执行的，而坐标变换是在此之后执行的。坐标变换的具体效果请见 7.3.1 节。

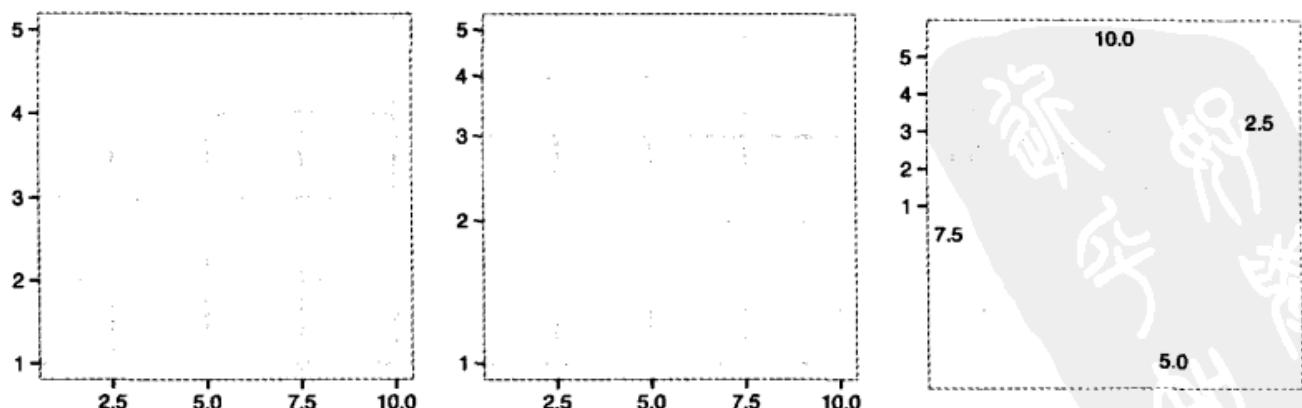


图 3.9 三种不同坐标系的坐标轴和网格线：笛卡尔 (Cartesian)、半对数 (semi-log) 和极坐标系 (polar)。极坐标系展示了非笛卡尔坐标系的缺点：很难画好坐标轴。

坐标系控制着坐标轴和网格线的绘制。图 3.9 列出了三种不同的坐标系。笛卡尔坐标系的绘图帮助文档非常少，所以要想在该类坐标系中绘制出精美的图形，需要投入更多的精力。7.3 节给出了坐标系更详细的介绍。

3.5.4 分面

分面在绘图时也非常有用，因此我们要把它加到图形的基本框架中。分面是条件绘图 (conditioned plots) 和网格绘图 (trellised plots) 的一般形式，通过它你可以方便地展示数据的不同子集。特别是当验证在不同条件下模型是否保持一致时，分面绘图是一个非常强大的工具。分面可以设定哪些变量可用来分割数据，以及设定是否应该对位置标度加以限制。关于分面的详细内容请见第 7 章。

3.6 数据结构

`ggplot2` 的图形语法通过一种非常简单直接的方法编码到 R 的数据结构中。一个图形对象就是一个包含 **数据**、**映射** (默认的图形属性映射)、**图层**、**标度**、**坐标**和**分面**的列表。图形对象还有一个目前我们还未讨论的组件：**options**。它专门用来存储特定的图形主题选项，详见第 8 章。

绘图有两种方式：一种是“一步到位”式，即利用前面学过的 `qplot()`；另一种是“逐层叠加”式，即利用 `ggplot()` 函数和图层函数逐步作图，细节请见第 4 章。当我们得到一个图形对象时，可以对它进行如下处理：

- 用 `print()` 函数将其呈现到屏幕上。在交互式操作时 `print()` 会自动被调用，但是在循环或函数里，我们需要手动输入 `print()`；
- 用 `ggsave()` 函数将其保存到磁盘，详见 8.3 节；
- 用 `summary()` 简单查看它的结构；
- 用 `save()` 函数把它的缓存副本保存到磁盘；这样可以保存一个图形对象的完整副本，你可以调用 `load()` 函数来重现该图。注意数据是储存在图形对象里的，所以如果你在图形对象外修改数据，然后重新读入已保存的图形对象，图像将不会更新。

下面的代码给出了这些工具的一些具体应用：

```
1 > p <- qplot(displ, hwy, data = mpg, colour = factor(cyl))
2 > summary(p)
3 data: manufacturer, model, displ, year, cyl, trans, drv, cty, hwy, fl,
4   class [234x11]
5 mapping: colour = factor(cyl), x = displ, y = hwy
6 facetting: facet_null()
7 -----
8 geom_point:
9 stat_identity:
10 position_identity: (width = NULL, height = NULL)
11
12 > ## 保存图形对象
13 > save(p, file = "plot.rdata")
14 > ## 读入图形对象
15 > load("plot.rdata")
16 > ## 将图片保存成 png 格式
17 > ggsave("plot.png", width = 5, height = 5)
```

第 4 章 用图层构建图像

4.1 简介

`qplot()` 的局限性在于它只能使用一个数据集和一组图形属性映射，解决这个问题的办法就是使用图层。每个图层可以有自己的数据集和图形属性映射，附加的数据元素可通过图层添加到图形中。

本章主要对图层、几何对象、统计变换和位置调整的工作方式作一些技术说明：如何对它们进行调用和自定义。下一章“工具箱”主要介绍怎样利用不同的几何对象和统计变换来解决特定的可视化问题。这两章紧密相连，本章介绍理论，下一章则介绍图层的应用。

4.2 节将教你如何手动创建一个图形对象，在这之前都是 `qplot()` 为我们完成这项任务。图形最初不会显示，直到至少一个图层被添加后图形才会显示出来，相关内容见 4.3 节。该节对图层进行了详细的说明并且给出了一些简洁的作图方法。在前面的章节里我们已经介绍过，一个图层由五个部分组成：

- 数据，必须是一个数据框 (data frame)，在绘图结束后可以被修改。详见 § 4.4；
- 一组图形属性映射，用来设定数据集中的变量如何映射到该图层的图形属性，详见 § 4.5。该节涵盖了三方面内容：通过图层设定 (setting) 来改变默认绘图参数的方法，设定和映射之间的区别，以及重要的分组图形属性；
- 几何对象，用来指定在图层中用哪种几何对象来绘图。几何对象决定了一组可用的图形属性，详见 § 4.6；
- 统计变换，对原数据做一些有用的统计变换。统计变换返回一个包含新变量的数据框，这些新变量也可通过特殊的语法映射到图形属性中。详见 § 4.7；

- 位置调整，通过调整元素位置来避免图形重合。详见 § 4.8。

最后，4.9 节对本章内容做了总结，讲解了如何结合本章所学的内容来绘制新颖的可视化图形，以及通过图形来挖掘数据所蕴涵的信息的方法。

4.2 创建绘图对象

当我们调用 `qplot()` 时，它其实为我们做了很多幕后工作：创建一个图形对象，添加图层并且展示结果。在整个过程中它使用了很多默认的绘图参数。如果想手动创建图形对象，就要用到 `ggplot()` 函数。该函数有两个主要的参数：**数据**和**图形属性映射**。这两个参数将被设为绘图的默认参数，只有在新添加的图层里设定了新参数时，默认值才会被修改。参数**数据**指定绘图所用的默认数据集（必须是数据框）；参数**映射**的设定方法与前面讲过的 `qplot()` 非常相似，只需要将图形属性和变量名放到函数 `aes()` 的括号里面即可。关于函数 `aes()` 的更多细节见 4.5 节，使用非常简单。下面的例子设定了一组默认映射：`x` 为 `carat`, `y` 为 `price`, `colour` 为 `cut`.

```
1 p <- ggplot(diamonds, aes(carat, price, colour = cut))
```

这个图形对象在加上图层之前无法显示，因此你现在什么也看不见。

4.3 图层

最简易的图层莫过于只设定一个几何对象，这是数据可视化的一种方法。如果给图形对象添加一个点几何对象（point geom）就会得到一个散点图。

```
1 p <- p + layer(geom = "point")
```

注意我们是如何用 `+` 来添加图层的。该图层使用了默认的数据集和图形属性映射，并且使用了另外两个可选参数的默认值：统计变换和位置调整。下面的图层设定列出了更多可选的参数选项。

```
1 layer(geom, geom_params, stat, stat_params, data, mapping, position)
```

下面给出了一个更复杂的代码。该代码可以生成一个组距为 2、铁青色的直方图。

```

1 p <- ggplot(diamonds, aes(x = carat))
2 p <- p + layer(
3   geom = "bar",
4   geom_params = list(fill = "steelblue"),
5   stat = "bin",
6   stat_params = list(binwidth = 2)
7 )
8 p

```

该图层的参数设定得非常细致但是过于繁琐，我们可以用**快捷函数** (shortcut) 来简化上面的代码。因为每一个几何对象都对应着一个默认的统计变换和位置参数，而每一个统计变换都对应着一个默认的几何对象参数，所以对于一个图层我们只需要设定 `stat` 或 `geom` 参数即可。下面的代码将生成与上述代码完全相同的图层。

```
1 geom_histogram(binwidth = 2, fill = "steelblue")
```

所有这类快捷函数都有相同的形式——以 `geom_` 或者 `stat_` 开头^①：

```

1 geom_XXX(mapping, data, ..., stat, position)
2 stat_XXX(mapping, data, ..., geom, position)

```

它们的参数定义了图层的各种组件：

- **mapping** (可选)：一组图形属性映射，通过 `aes()` 函数来设定，详见 4.5 节；
- **data** (可选)：一个数据集，它会修改默认的数据集。大部分情况下该参数被省略掉，默认数据集将被调用。详见 4.4 节；
- ...: **geom** 或 **stat** 的参数，例如直方图的组距 (`binwidth`) 或者 loess 光滑曲线的带宽 (`bandwidth`)。我们也可以用图形属性作为参数，这样该属性就被设定为一个固定的值，而不是被映射给数据集中的一个变量。上面的例子展示了如何将直方图的填充色设为“铁青色”。4.5.2 节给出了更多的例子；
- **geom** 或 **stat** (可选)：我们可以修改 `geom` 默认的 `stat` 值，或者 `stat` 所默认的 `geom` 值。它们是一组字符串，包含了将要使用的几何对象或统计

^①译者注：原书此处代码有误。

变换的名称^②。使用默认值将会得到标准的图形；修改默认值会得到一些新奇的图形，详见 4.9.1 节；

- **position** (可选)：选择一种调整对象重合的方式，详见 4.8 节。

注意，参数 `data` 和 `mapping` 在 `ggplot()` 函数和图层函数中的位置是相反的。这是因为我们在图形对象中一般先设定数据集，而在图层函数中大多是设定图形属性而不是数据集。我建议尽量写清参数名而不要依赖参数的相对位置来设定参数，这样可以使得代码更具可读性，这也是本书遵循的原则之一。

图层可以被添加到用 `ggplot()` 或 `qplot()` 创建的图形对象上。实际上 `qplot()` 的绘图原理就是先创建图形对象然后再添加图层。下面的例子展示了这两种绘图方法的等价性。

```
1 ggplot(msleep, aes(sleep_rem / sleep_total, awake)) + geom_point()
2 ## 等价于
3 qplot(sleep_rem / sleep_total, awake, data = msleep)
4
5 ## 也可以给 qplot 添加图层
6 qplot(sleep_rem / sleep_total, awake, data = msleep) + geom_smooth()
7 ## 等价于
8 qplot(sleep_rem / sleep_total, awake, data = msleep,
9       geom = c("point", "smooth"))
10 ## 或
11 ggplot(msleep, aes(sleep_rem / sleep_total, awake)) +
12   geom_point() + geom_smooth()
```

你已经注意到图形对象可以存储到一个变量里。`summary` 函数可以帮助我们查看图形对象的结构而不用直接绘制出图形，如下例所示。`summary` 首先给出图形对象的默认设置，然后给出每个图层的信息。标度和分面将在第 6 章和第 7 章学习。

```
1 > p <- ggplot(msleep, aes(sleep_rem / sleep_total, awake))
2 > summary(p)
3 data: name, genus, vore, order, conservation, sleep_total, sleep_rem,
4   sleep_cycle, awake, brainwt, bodywt [83x11]
5 mapping: x = sleep_rem/sleep_total, y = awake
6 facetting: facet_null()
```

^②译者注：此处原书只写了几何对象，没有写统计变换，译者认为应该加上。

```

7 >
8 > p <- p + geom_point()
9 > summary(p)
10 data: name, genus, vore, order, conservation, sleep_total, sleep_rem,
11   sleep_cycle, awake, brainwt, bodywt [83x11]
12 mapping: x = sleep_rem/sleep_total, y = awake
13 facetting: facet_null()
14 -----
15 geom_point: na.rm = FALSE
16 stat_identity:
17 position_identity: (width = NULL, height = NULL)

```

图层是普通的 R 对象，所以可以存储到变量里去，这有利于代码避繁就简。例如，一组图形可以先用不同的数据来进行初始化，然后加上相同的图层，如果后面想改变图层，只需要修改一个地方即可。下面的例子创建了一个带有半透明深蓝色回归线的图层^③。

```

1 library(scales)
2 bestfit <- geom_smooth(method = "lm", se = F,
3   colour = alpha("steelblue", 0.5), size = 2)
4 qplot(sleep_rem, sleep_total, data = msleep) + bestfit
5 qplot(awake, brainwt, data = msleep, log = "y") + bestfit
6 qplot(bodywt, brainwt, data = msleep, log = "xy") + bestfit

```

下节将详细介绍数据和映射，然后继续介绍可用的几何对象、统计变换和位置调整。

4.4 数据

ggplot2 对于数据集的要求很简单：必须是一个数据框 (data frame)。这里对数据集的限制与 R 中其他绘图包的要求不同，例如 Lattice 函数可以选择性地指定数据框，或者直接利用全局环境里的向量。然而 ggplot2 对数据集的这种限制是有充分理由的。数据非常重要，我们最好要明确对数据都作了怎样的处理。ggplot2 会从我们给定的数据框中提取绘图所需要的变量，并生成一个新的数据集，而不是直接在原数据上进行数据变换。关于数据变换的问题，你可以

^③译者注：0.9.1 及之后版本需要加载 scales 包。

参考 `plyr` 和 `reshape`(新出的 `reshape2`) 这两个非常有用的软件包。由于保存一个数据框要比保存多个向量方便得多，所以无论是向别人发送数据还是让别人重复运行自己的代码，使用数据框都会更方便。

这种限制也可以方便我们用相同的代码、不同的数据集绘图——只需要改变数据集即可。如下例所示，用 `%+%` 来添加新的数据集以代替原来的数据集(你可能会感到奇怪，为什么不像添加其他的图形元素一样用 `+` 呢？因为 R 语言的限制使得不能对数据直接用 `+`。)

```
1 p <- ggplot(mtcars, aes(mpg, wt, colour = cyl)) + geom_point()
2 p
3 mtcars <- transform(mtcars, mpg = mpg ^ 2)
4 p %+% mtcars
```

在更改数据集时，可以任意改变它的值和维数，但是如果将一个变量从离散型变成连续型或者从连续型变成离散型，那么也需要改变相应的默认标度^④，详见 6.3 节。

在不使用分面的时候不必设定默认的数据集；分面是一个全局操作(作用于所有的图层)，并且它需要一个定义了分面变量的默认数据集，详见 § 7.2.4。如果没有给定默认的数据集，那么每个图层都要设定自己的数据集。

数据是以副本而不是引用的形式存储到图形对象中的。这样做有两个重要的好处：其一，如果你的数据改变了，绘图不会改变。其二，`ggplot2` 的对象都是自含型的 (self-contained)，所以它们可以被存储 (`save()`) 到磁盘上，并且之后可以被直接加载运行 (`load()`)。

4.5 图形属性映射

`aes()` 函数用来将数据变量映射到图形中，从而使变量成为可以被感知的图形属性。`aes()` 函数里有一系列的图形属性参数：

```
1 aes(x = weight, y = height, colour = age)
```

这里我们将 `x` 坐标映射到 `weight`，`y` 坐标映射到 `height`，`colour` 映射到 `age`。前两个参数可以省略名字，`weight` 和 `height` 会自动匹配到 `x` 和 `y`，这

^④译者注：在 0.9.1 及之后版本中不需要担心变量变换时标度的更改。

和 `qplot()` 的用法是一致的。注意，你最好不要使用指定数据集以外的变量（例如 `diamonds$carat`），因为这样无法将绘图所用的数据都封装到一个对象里。

```
1 aes(weight, height, colour = sqrt(age))
```

注意也可以使用变量的函数值作为参数。

每个 `aes()` 函数里的变量都必须包含于默认数据集或者图层数据集中，这是保证 `ggplot2` 对象都是自含型的重要方式之一，这样方便存储和重复使用。

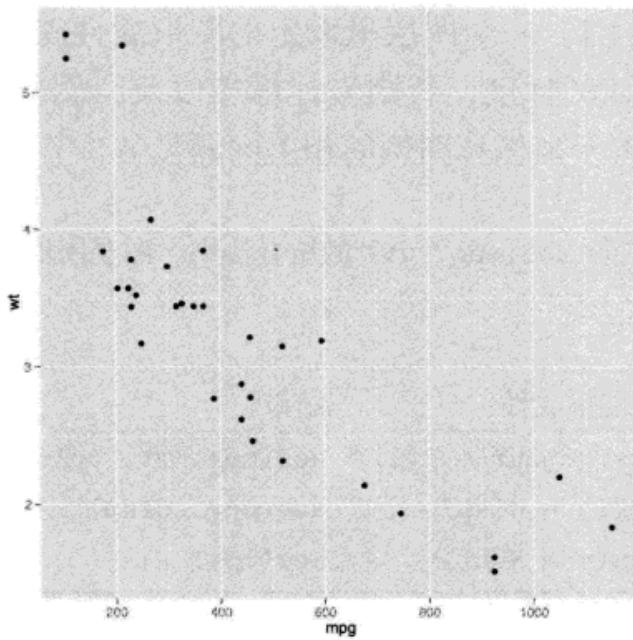
4.5.1 图和图层

如下例所示，默认的图形属性映射可以在图形对象初始化时设定，或者过后用 `+` 修改。

```
1 > p <- ggplot(mtcars)
2 > summary(p)
3 data: mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb [32x11]
4 facetting: facet_null()
5 >
6 > p <- p + aes(wt, hp)
7 > summary(p)
8 data: mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb [32x11]
9 mapping: x = wt, y = hp
10 facetting: facet_null()
```

用 `+` 修改图形属性映射的一个例子见 4.9.3 节。到目前为止，我们已经见过好几个用默认的参数映射来添加图层的例子：

```
1 > p <- ggplot(mtcars, aes(x = mpg, y = wt))
2 > p + geom_point()
```



如下面代码所示，图形对象 p 中默认的映射可以在新图层里进行扩充或修改，结果见图 4.1。

```
1 p + geom_point(aes(colour = factor(cyl)))
2 p + geom_point(aes(y = disp))
```

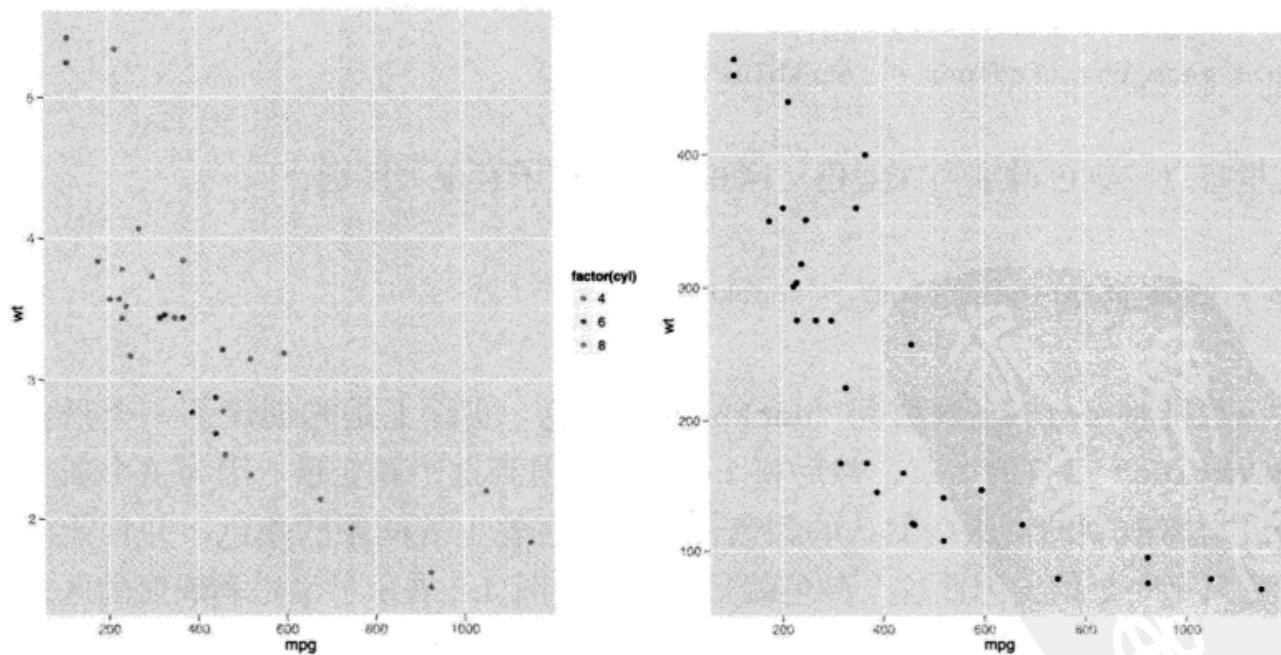


图 4.1 (参见彩图 8) 修改图形属性。用 `factor(cyl)` 修改颜色 (左)，用 `disp` 修改 y 坐标 (右)。

具体的规则见表 4.1。一个图层里设定的图形属性映射只对该图层起作用。因此，除非你修改默认的标度，否则坐标标签以及图例标题都会根据图形对象的默认设置而生成。关于坐标和图例的相关问题见 6.5 节。

表 4.1 图层映射和默认映射 `aes(mpg, wt)` 的结合法则。图层图形属性可以添加、修改和删除默认映射。

操作	层图形属性	结果
添加	<code>aes(colour = cyl)</code>	<code>aes(mpg, wt, colour = cyl)</code>
修改	<code>aes(y = disp)</code>	<code>aes(mpg, disp)</code>
删除	<code>aes(y = NULL)</code>	<code>aes(mpg)</code>

4.5.2 设定和映射

除了可以将一个图形属性映射到一个变量，你也可以在图层的参数里将其设定为一个单一值（例如，`colour = "red"`）。图形属性可以根据观测的不同而变化，但是参数则不行。下面的例子用图层里的 `colour` 参数设定了点的颜色：

```
1 p <- ggplot(mtcars, aes(mpg, wt))
2 p + geom_point(colour = "darkblue")
```

这里将点的颜色设定为深蓝色。这和下面的例子有很大区别：

```
1 p + geom_point(aes(colour = "darkblue"))
```

这里将 `colour` 映射到 "darkblue" 颜色。实际上是先创建了一个只含有 "darkblue" 字符的变量，然后将 `colour` 映射到这个新变量。因为这个新变量的值是离散型的，所以默认的颜色标度将用色轮上等间距的颜色，并且此处新变量只有一个值，因此这个颜色就是桃红色。图 4.2 展示了设定和映射的区别。

在使用 `qplot()` 函数的时候，可以将某个值放到 `I()` 里来实现映射（例如，`colour = I("darkblue")`）。附录 B 介绍了如何为各种各样的图形属性设定合适的值。

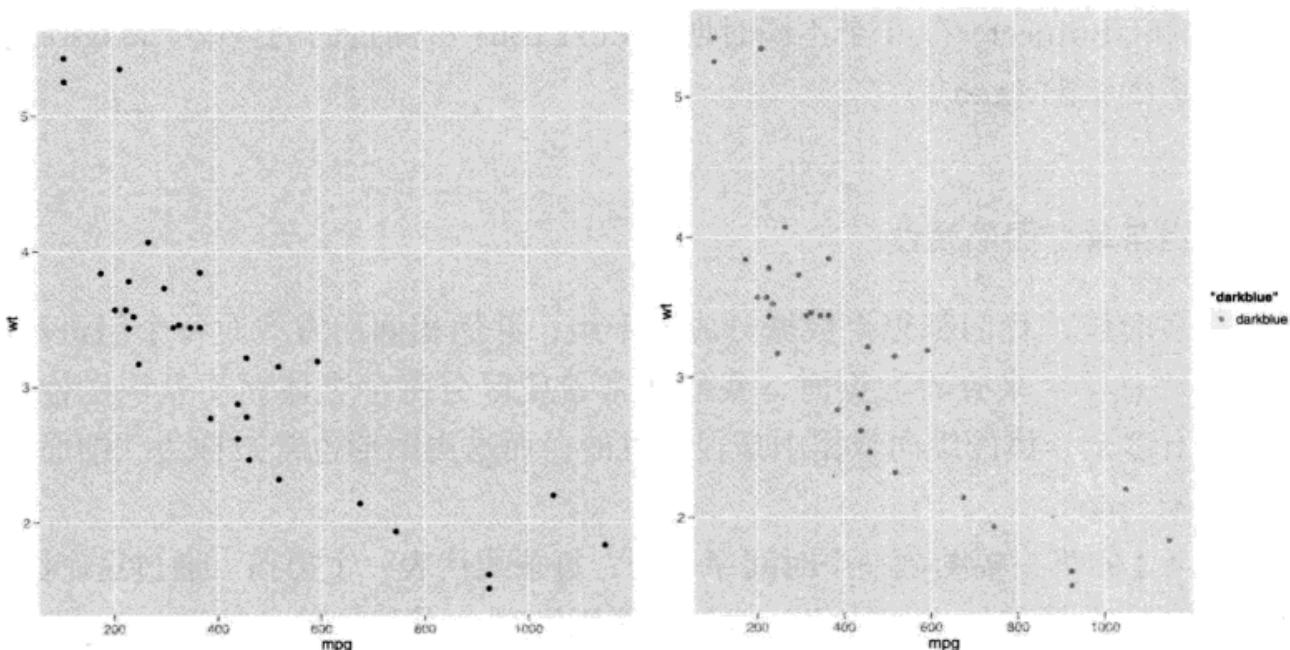


图 4.2 (参见彩图 9) 将颜色设定为 "darkblue" (左) 与将颜色映射到 "darkblue" (右) 的区别。当颜色映射到 "darkblue" 时, "darkblue" 将被看作一个普通的字符串, 使用默认的颜色标度进行标度转换, 结果得到了粉红色的点和图例。

4.5.3 分组

在 ggplot2 里, 几何对象可以大致分为个体 (individual) 几何对象和群组 (collective) 几何对象两大类。个体几何对象对数据框的每一条数据绘制一个可以区别于其他个体的图形对象。例如, 点几何对象用点来表示每一条观测。相反地, 群组几何对象用来表示多条观测, 它们可以是某个统计摘要的一个结果, 或者是几何对象的基础展示, 例如多边形。线条和路径介于这两者之间: 每条线都由许多线段组成, 而每条线段又代表两个点。我们该如何来控制哪些观测值用哪种图形元素呢? 这就是分组 (group) 图形属性的工作。

图中所有离散型变量的交互作用被设为分组的默认值, 通常这样可以正确地给数据分组, 但是如果没能正确分组或者图中没有离散型变量, 那么就需要自定义分组结构, 即将 group 映射到一个在不同的组有不同取值的变量。当现有的单个变量不能够正确地分组, 而两个变量的组合可以正确分组时, 可以使用 interaction() 函数。

通常有三种情况是默认分组不能解决的, 下面我们将一一介绍。在下面的例子里, 我们将用 nlme 包里的一个简单的纵向数据集 Oxboys。该数据记录了

26名男孩(Subject)在9个不同时期(Occasion)所测定的身高(height)和中心化后的年龄(age)。

多个分组与单个图形属性

很多时候，我们都想将数据分成若干组，并用相同的方式对每个组进行渲染。当从总体上来查看数据时，我们通常希望区分每个个体而不是识别他们。这在含有多个个体的纵向数据中是很常见的，而这类图形也常被称为“细面图”(spaghetti plot)。

图4.3的第一幅图是一个时间序列图，每条线代表一个男孩。通过这个图你可以看到每个男孩的成长轨迹，但是不能识别哪个男孩是哪条轨迹。生成这个图的代码是：

```
1 p <- ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
```

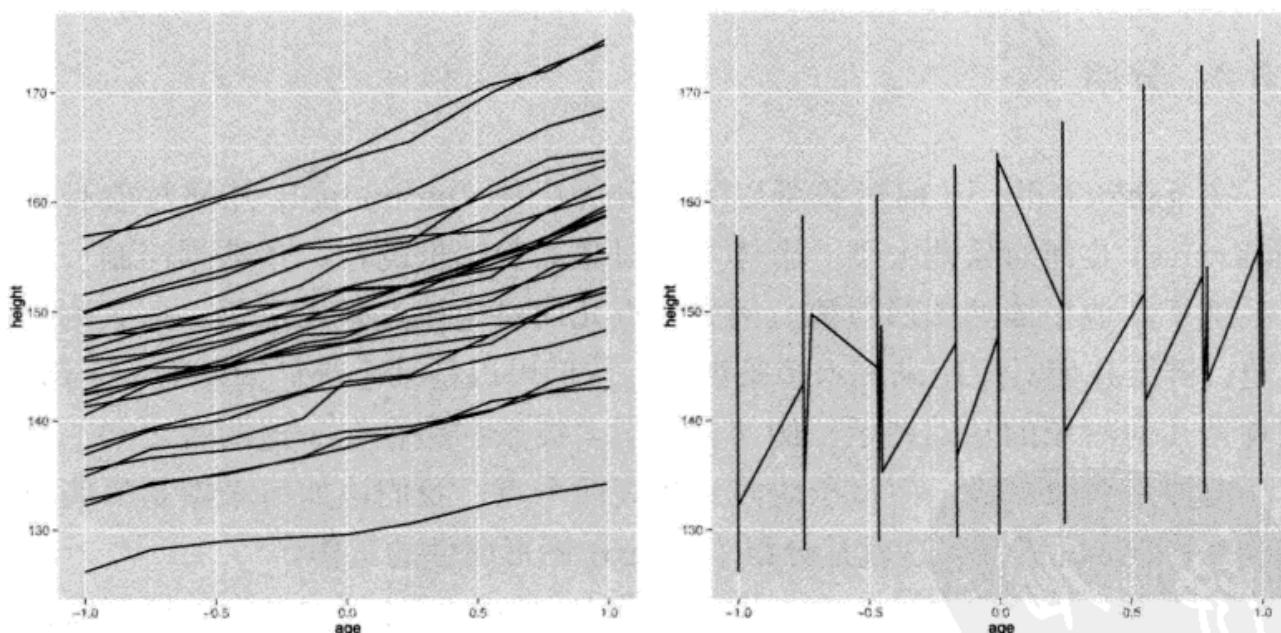


图4.3 正确分组时(分组变量`group = Subject`)每个个体的折线图(左)。错误分组时连接所有观测点的折线图(右)。此处省略了分组图形属性，效果等同于`group = 1`。

我们指定`Subject`为分组变量，每个男孩对应一条线。第二幅图没有指定分组变量，我们得到了一条通过每一个点的单一线条，从图中可以看到，这个图几乎毫无用处！这是典型的因为没有正确分组而生成的奇怪折线图。

不同图层上的不同分组

有时我们想根据不同水平下的数据整合来对统计汇总信息 (summary) 进行图形绘制，从而不同的图层可能有不同的分组图形属性，因此，有的图层展示个体水平的数据，而有的图层则展示更大组群的统计信息。

在前面例子的基础上，假设我们想根据所有男孩的年龄和身高在图中添加一条光滑线条。如果还用和前面绘制折线图时同样的分组方式，我们将会得到图 4.4 的左图。

```
1 p + geom_smooth(aes(group = Subject), method = "lm", se = F)
```

这并不是我们想要的结果；我们无意间给每一个男孩添加了一条光滑线条。因此，新图层需要一个不同的分组图形属性，`group = 1`，这样所绘出的线条才是基于整体数据的，效果见图 4.4 的右图。修改后的代码如下：

```
1 p + geom_smooth(aes(group = 1), method = "lm", size = 2, se = F)
```

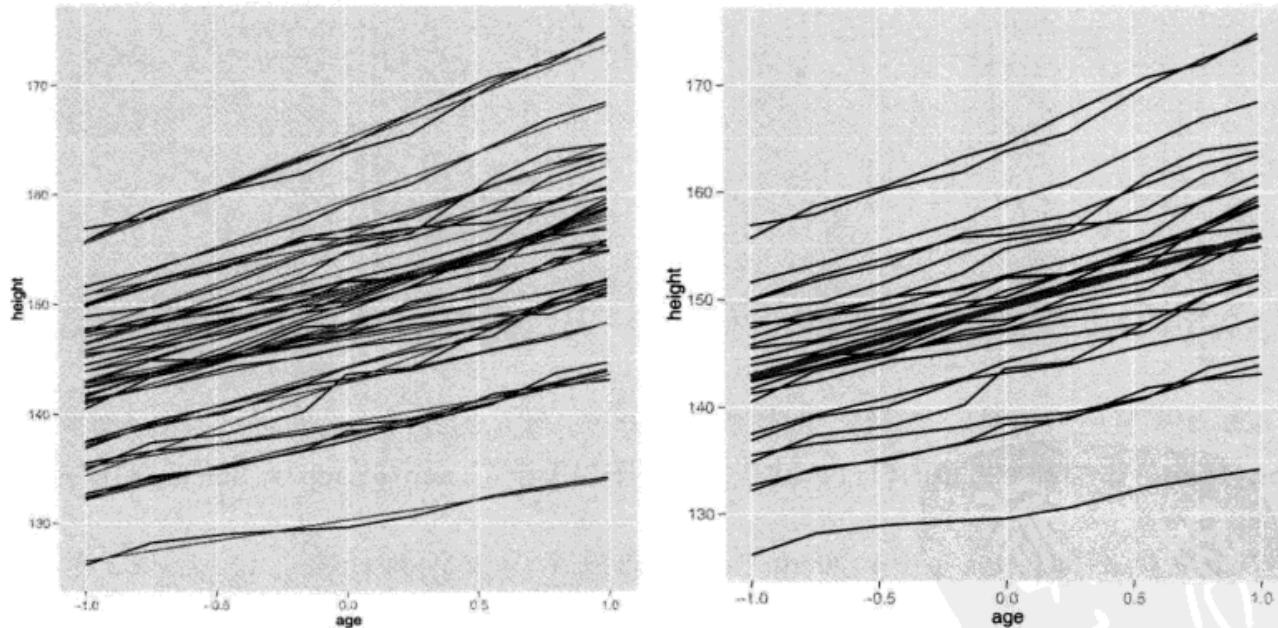


图 4.4 给 Oxboys 数据添加光滑曲线。左图用了和折线图同样的分组变量，得到了每个男孩的拟合直线。右图在平滑层里用了 `aes(group = 1)`，得到了所有男孩的拟合直线。

注意，我们已经将第一幅图存储到变量 `p` 中了，我们可以在此基础上添加第二个图层，而不需要重新输入第一个图层的代码。这样可以有效节省时间，更多细节请见第 10 章。

修改默认分组

如果图像中含有离散型变量，而你却想绘制连接所有分组的线条，那么你可以采取绘制交互作用图、轮廓图以及平行坐标图时所用的策略。这里以绘制各个测量时期 (Occasion) 身高 (height) 的箱线图为例，如图 4.5 的左图所示。

```
1 boysbox <- ggplot(Oxboys, aes(Occasion, height)) + geom_boxplot()
```

这里没有必要设定组图形属性；因为 Occasion 是一个离散型变量，所以默认的分组变量就是 Occasion。要在此基础上添加个体轨迹，我们需要用 `aes(group = Subject)` 修改第一层的默认分组，如图 4.5 的右图所示。

```
1 boysbox + geom_line(aes(group = Subject), colour = "#3366FF")
```

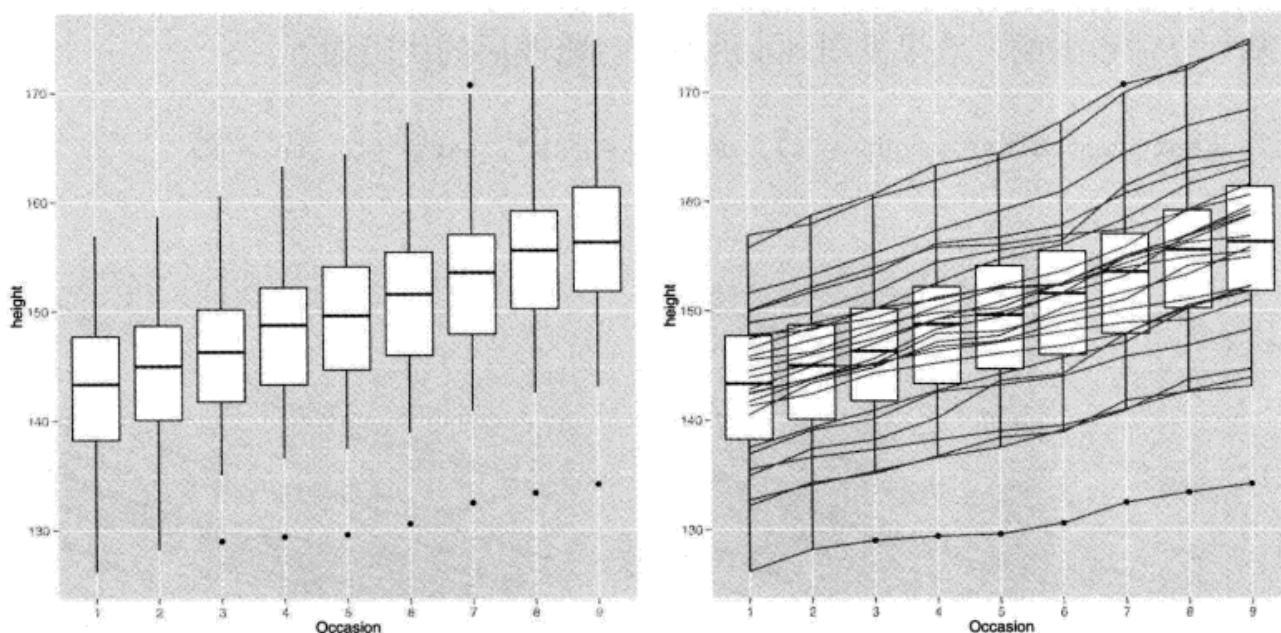


图 4.5 如果想用箱线图来查看每个时期的身高分布，默认的分组是正确的（左图）。如果想用 `geom_line()` 添加每个男孩的轨迹，就需要在新图层里设定 `aes(group = Subject)`（右图）。

在第二个图层中，我们改变了线条的颜色以区别箱线图。这又是一个将某个图形属性设定为固定值的例子。此处颜色是一个渲染属性，在数据中没有相对应的变量。

4.5.4 匹配图形属性和图形对象

群组几何对象另一重要议题是，如何将个体的图形属性映射给整体的图形属性。对于个体几何对象而言这不是个问题，因为每一条观测都被一个单一的

图形元素所表示。然而，高密度数据将会使得区别单个的点变得困难（或者不可能），这也意味着如果单个点几何对象变成一个群组几何对象，就变成了一大团点。

线条和路径遵循差一原则：观测点比线段数目多一，第一条线段将使用第一条观测的图形属性，第二条线段将使用第二条观测的图形属性，依次类推。这意味着最后一条观测的图形属性将不会被用到，如图 4.6 所示。路径和线条的另一个限制是每个个体的线条类型必须是一个常数，因为 R 不能绘制不同线条类型相连接的线条。

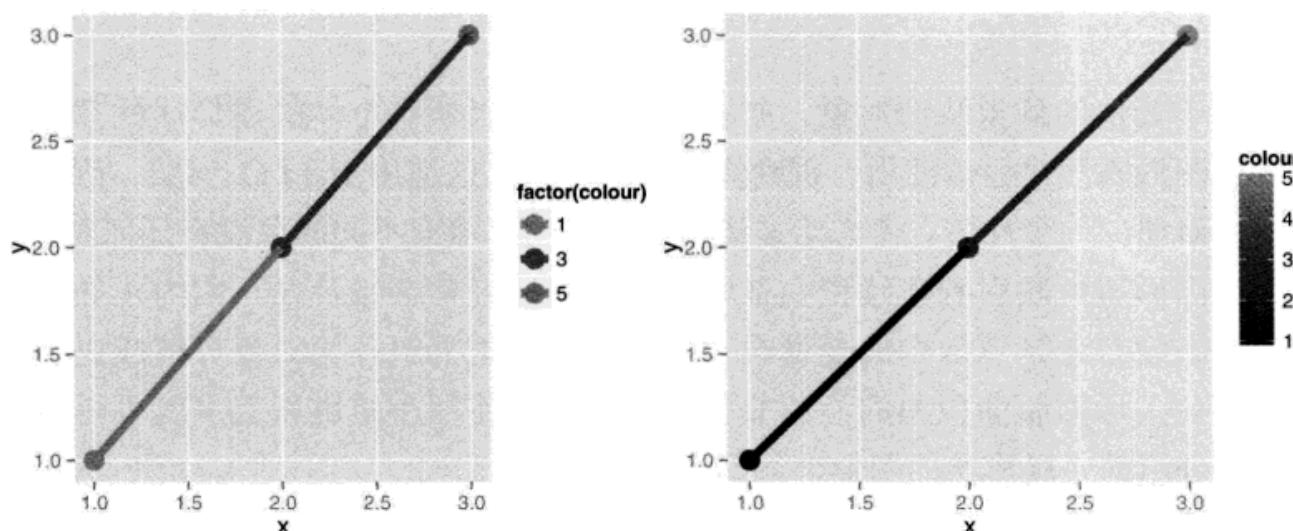


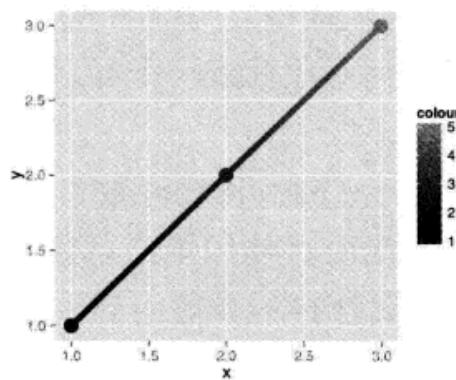
图 4.6 (参见彩图 10) 对于线条和路径，线段的图形属性是由起始点的图形属性决定的。如果颜色是离散的 (左图)，在相邻的颜色间插入其他颜色是没有任何意义的。如果颜色是连续的 (右图)，可以在相邻的颜色间进行插补，但默认条件下 R 不会这样做。

你还可以想象一个更为复杂的系统，其中的线段平稳地从一种图形属性变换到另一种图形属性。这种方式对连续型变量很有效，如大小和颜色，但并不适用于渐变的线条类型。如果想得到这种效果可以用线性插值法来做，代码如下：

```

1 > xgrid <- with(df, seq(min(x), max(x), length = 50))
2 > interp <- data.frame(
3 +   x = xgrid,
4 +   y = approx(df$x, df$y, xout = xgrid)$y,
5 +   colour = approx(df$x, df$colour, xout = xgrid)$y
6 + )
7 > qplot(x, y, data = df, colour = colour, size = I(5)) +
8 +   geom_line(data = interp, size = 2)

```



对于其他的群组几何对象，如多边形，只有当所有个体的图形属性都相同时，该图形属性才会被使用，否则将使用默认值。这适用于 `fill` 参数，因为它是整体对象的一个性质：考虑给多边形边界上的点填充不同的色没有任何意义。

当图形属性映射到连续型变量时，这类问题经常会被涉及。因为正如上所述，当映射对象是离散型的变量时，它将默认地把群组几何对象分解成更小的块。这对条形图和面积图的绘制非常有效，因为将每小块堆积起来就可得到和原来未分组前一样形状的图形，如图 4.7 所示。

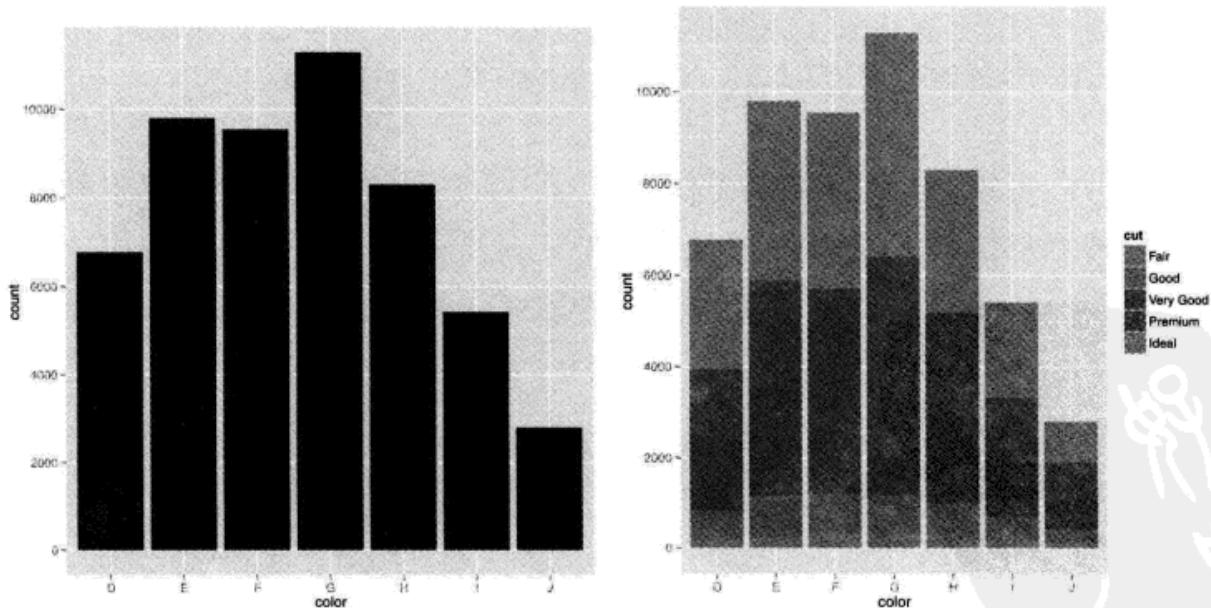


图 4.7 (参见彩图 11) 一个条形图 (左) 按组分解后得到的叠加条形图 (右)，两者轮廓相同。

4.6 几何对象

几何图形对象，简称为 `geom`，它执行着图层的实际渲染，控制着生成的图像类型。例如，用点几何对象 (`point geom`) 将会生成散点图，而用线几何对象 (`line geom`) 会生成折线图。表 4.2 列出了 `ggplot2` 里所有可用的几何对象^⑤。

每个几何对象都有一组它能识别的图形属性和一组绘图所需的值。例如，一个点含有颜色、大小和形状等图形属性，以及 `x` 和 `y` 位置坐标。一个条形含有高度、条宽、边界颜色和填充颜色等图形属性值。表 4.3 列出了所有几何对象的图形属性值。

有些几何对象主要在它们参数化的方式上有所不同。例如，瓦块几何对象 (`tile geom`) 设定的是其中心位置、长和宽，而矩形几何对象 (`rect geom`) 设定的是它的上 (`ymax`)、下 (`ymin`)、左 (`xmin`) 和右 (`right`) 位置。实际上，矩形几何对象被看作一个多边形，它的参数是四个角的位置。这在非笛卡尔坐标系中是非常有用的，在第 7 章我们会详细介绍。

每一个几何对象都有一个默认的统计变换，并且每一个统计变换都有一个默认的几何对象。例如封箱 (`bin`) 统计变换默认使用条状几何对象 (`bar geom`) 来绘制直方图。这些默认值可参见表 4.3。修改这些默认值虽然能生成符合语法的绘图，但是它们可能会违反一些绘图惯例。见 4.9.1 节中的例子。

^⑤译者注：该表以 0.9.1 版本中的几何对象为准，于原书略有不同。

表 4.2 ggplot2 中的几何对象。

名称	描述
abline	线, 由斜率和截距决定
area	面积图 (area plot)
bar	条形图, 以 x 轴为底的矩形
bin2d	2 维热图
blank	空白, 什么也不画
boxplot	箱线图
contour	等高线图
crossbar	带有水平中心线的盒子图
density	光滑密度曲线图
density2d	二维密度等高线图
dotplot	“点直方图”, 用点来表示观测值的个数
errorbar	误差棒
errorbarh	水平的误差棒
freqpoly	频率多边形图
hex	用六边形表示的 2 维热图
histogram	直方图
hline	水平线
jitter	给点添加扰动, 减轻图形重叠问题
line	按照 x 坐标的大小顺序依次连接各个观测值
linerange	一条代表一个区间的竖直线
map	基准地图里的多边形
path	按数据的原始顺序连接各个观测值
point	点, 用来绘制散点图
pointrange	用一条中间带点的竖直线代表一个区间
polygon	多边形, 相当于一个有填充的路径
quantile	添加分位数回归线
raster	高效的矩形瓦片图
rect	2 维的矩形图
ribbon	色带图, 连续的 x 值所对应的 y 的范围
rug	边际地毯图
segment	添加线段或箭头
smooth	添加光滑的条件均值线
step	以阶梯形式连接各个观测值
text	文本注释
tile	瓦片图
violin	小提琴图
vline	竖直线



表 4.3 默认的统计变换和图形属性。黑体图形属性是必须声明的参数。

名称	默认的统计变换	图形属性
abline	abline	colour, linetype, size
area	identity	colour, fill, linetype, size, x , y
bar	bin	colour, fill, linetype, size, weight, x
bin2d	bin2d	colour, fill, linetype, size, weight, xmax , xmin , ymax , ymin
blank	identity	
boxplot	boxplot	colour, fill, lower , middle , size, upper , weight, x , ymax , ymin
contour	contour	colour, linetype, size, weight, x , y
crossbar	identity	colour, fill, linetype, size, x , y , ymax , ymin
density	density	colour, fill, linetype, size, weight, x , y
density2d	density2d	colour, linetype, size, weight, x , y
dotplot	bindot	colour, fill, x , y
errorbar	identity	colour, linetype, size, width, x , ymax , ymin
errobarh	identity	colour, linetype, size, width, y , ymax , ymin
freqpoly	bin	colour, linetype, size
hex	binhex	colour, fill, size, x , y
histogram	bin	colour, fill, linetype, size, weight, x
hline	hline	colour, linetype, size
jitter	identity	colour, fill, shape, size, x , y
line	identity	colour, linetype, size, x , y
linerange	identity	colour, linetype, size, x , ymax , ymin
map	identity	colour, fill, linetype, size, x , y , map_id
path	identity	colour, linetype, size, x , y
point	identity	colour, fill, shape, size, x , y
pointrange	identity	colour, fill, linetype, shape, size, x , y , ymax , ymin
polygon	identity	colour, fill, linetype, size, x , y
quantile	quantile	colour, linetype, size, weight, x , y
raster	identity	colour, fill, linetype, size, x , y
rect	identity	colour, fill, linetype, size, xmax , xmin , ymax , ymin
ribbon	identity	colour, fill, linetype, size, x , ymax , ymin
rug	identity	colour, linetype, size
segment	identity	colour, linetype, size, x , xend , y , yend
smooth	smooth	alpha, colour, fill, linetype, size, weight, x , y
step	identity	colour, linetype, size, x , y
text	identity	angle, colour, hjust, label , size, vjust, x , y
tile	identity	colour, fill, linetype, size, x , y
violin	ydensity	weight, colour, fill, size, linetype, x , y
vline	vline	colour, linetype, size

4.7 统计变换

统计变换，简称为 `stat`，即对数据进行统计变换，它通常以某种方式对数据信息进行汇总。例如，平滑 (smoother) 是一个很有用的统计变换，它能在一些限制条件的约束下计算给定 x 值时 y 的平均值。表 4.4 列出了目前可用的统计变换。为了阐明在图形中的意义，一个统计变换必须是一个位置尺度不变量，即 $f(x + a) = f(x) + a$ 并且 $f(b \cdot x) = b \cdot f(x)$ 。这样才能保证当改变图形的标度时，数据变换保持不变。

表 4.4 ggplot2 中的统计变换。

名称	描述
<code>bin</code>	计算封箱 (bin) 数据
<code>bin2d</code>	计算矩形封箱内的观测值个数
<code>bindot</code>	计算“点直方图”的封箱数据
<code>binhex</code>	计算六边形热图的封箱数据
<code>boxplot</code>	计算组成箱线图的各种元素值
<code>contour</code>	三维数据的等高线
<code>density</code>	一维密度估计
<code>density2d</code>	二维密度估计
<code>function</code>	添加新函数
<code>identity</code>	不对数据进行统计变换
<code>qq</code>	计算 qq 图的相关值
<code>quantile</code>	计算连续的分位数
<code>smooth</code>	添加光滑曲线
<code>spoke</code>	将角度和半径转换成 <code>xend</code> 和 <code>yend</code>
<code>sum</code>	计算每个单一值的频数，有助于解决散点图的图形重叠问题
<code>summary</code>	对每个 x 所对应的 y 值做统计描述
<code>summary2d</code>	对 2 维矩形封箱设定函数
<code>summaryhex</code>	对 2 维六边形封箱设定函数
<code>unique</code>	删除重复值
<code>ydensity</code>	小提琴图，计算 1 维 y 轴方向的核密度函数估计值

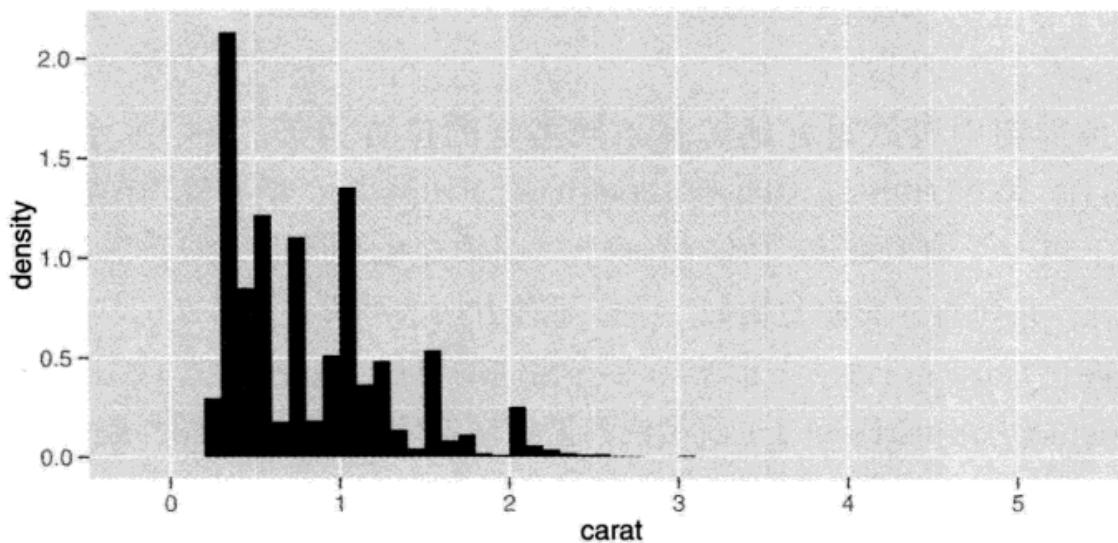
统计变换可将输入的数据集看做输入，将返回的数据集作为输出，因

此统计变换可以向原数据集中插入新的变量。例如，常被用来绘制直方图的 `stat_bin` 统计变换会生成如下变量：

- `count`, 每个组里观测值的数目；
- `density`, 每个组里观测值的密度 (占整体的百分数/组宽)；
- `x`, 组的中心位置。

这些生成变量 (generated variable) 可以被直接调用。例如，直方图默认将条形的高度赋值为观测值的频数 (`count`)，如果你更喜欢传统的直方图，你可以用密度 (`density`) 来代替。下面的例子给出了钻石数据集中的克拉 (`carat`) 的密度直方图。

```
1 > ggplot(diamonds, aes(carat)) +  
2 +   geom_histogram(aes(y = ..density..), binwidth = 0.1)
```



生成变量的名字必须要用 `..` 围起来。这样可以防止原数据集中的变量和生成变量重名时造成混淆，并且以后处理代码时，你可以很清晰得分辨出哪些变量是由统计变换生成的。每个统计变换的帮助文档里都列出了其生成变量的名称。

类似地，用 `qplot()` 生成该图像的代码如下：

```
1 qplot(carat, ..density.., data = diamonds, geom = "histogram",  
2       binwidth = 0.1)
```

4.8 位置调整

所谓位置调整，即对该层中的元素位置进行微调。表 4.5 列出了 `ggplot2` 中所有可用的位置调整参数。位置调整一般多见于处理离散型数据，连续型数据一般很少出现完全重叠的问题，当出现了这类问题时（由于数据密度高），即使微调，如随机扰动，通常也无法解决问题。

表 4.5 五种位置调整参数。

名称	描述
dodge	避免重叠，并排放置
fill	堆叠图形元素并将高度标准化为 1
identity	不做任何调整
jitter	给点添加扰动避免重合
stack	将图形元素堆叠起来

在条形图中可以很好地解释不同类型的位置调整。图 4.8 展示了堆叠 (stacking)、填充 (filling) 和并列 (dodging) 三种类型。堆叠就是在相同的 x 坐标位置上将一个条形堆砌到另一个的上方；填充也是如此，但条形的高度被标准化为 1；并列与分面非常相似。关于每种方法的利弊请见 7.2.6 节。要使这些操作都能正常运行，每个条形必须要有相同的宽度并且相互之间无重合。同一调整 (identity adjustment)(也就是说不做任何处理) 对于条形图不起作用，但作为参考，图 4.9 仍将它与用相同数据作出的折线图进行了比较。

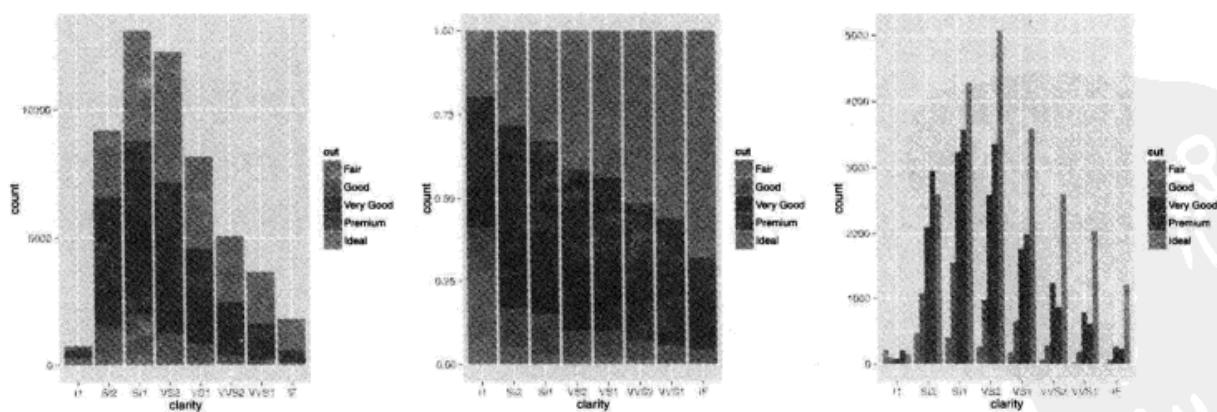


图 4.8 (参见彩图 12) 应用于条形图的三种位置调整。从左到右依次是：堆叠 (stacking)，填充 (filling) 和并列 (dodging)。

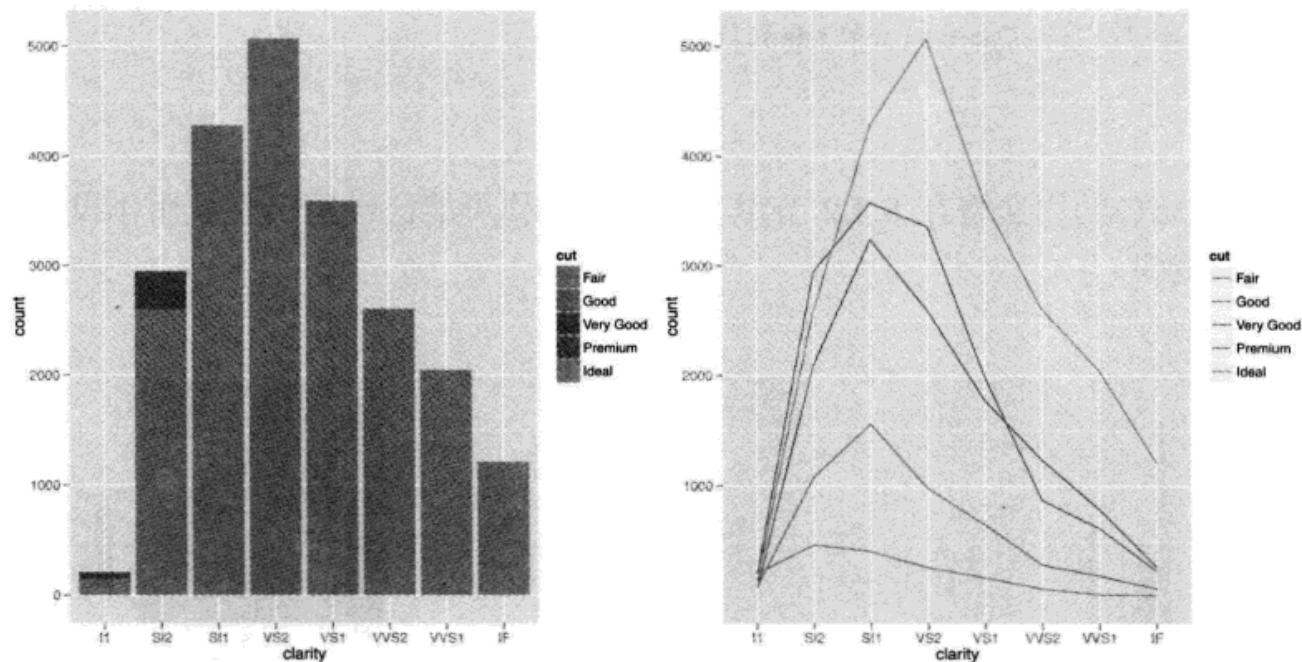


图 4.9 (参见彩图 13) 同一调整 (identity adjustment) 不适用于条形图 (左), 因为后画的条形会挡住先画的条形。但它适用于线型图 (右), 因为线条不存在相互遮掩的问题。

4.9 整合

一旦熟悉了图层操作, 就可以轻松作出复杂的图形。下面的例子将演示一些本章所介绍的图层功能。这些例子只是帮你起步, 要想绘制更精妙的图形则需要你发挥无穷的想象力了!

4.9.1 结合几何对象和统计变换

将几何对象和不同的统计变换进行组合, 你将可以轻松地绘制出新颖的图形。图 4.10 给出了直方图的三个“变体”。它们都应用了基于直方图的统计变换, 但使用了不同的几何对象来展示结果: 面积、点和瓦块 (tile)。

```

1 d <- ggplot(diamonds, aes(carat)) + xlim(0, 3)
2 d + stat_bin(aes(ymax = ..count..), binwidth = 0.1, geom = "area")
3 d + stat_bin(
4   aes(size = ..density..), binwidth = 0.1,
5   geom = "point", position = "identity"
6 )
7 d + stat_bin(
8   aes(y = 1, fill = ..count..), binwidth = 0.1,

```

```

9     geom = "tile", position="identity"
10 )

```

(`xlim()`的用法将在 6.4.2 节里讨论, 但是你可能已经猜出它被用作设定水平轴的范围。)

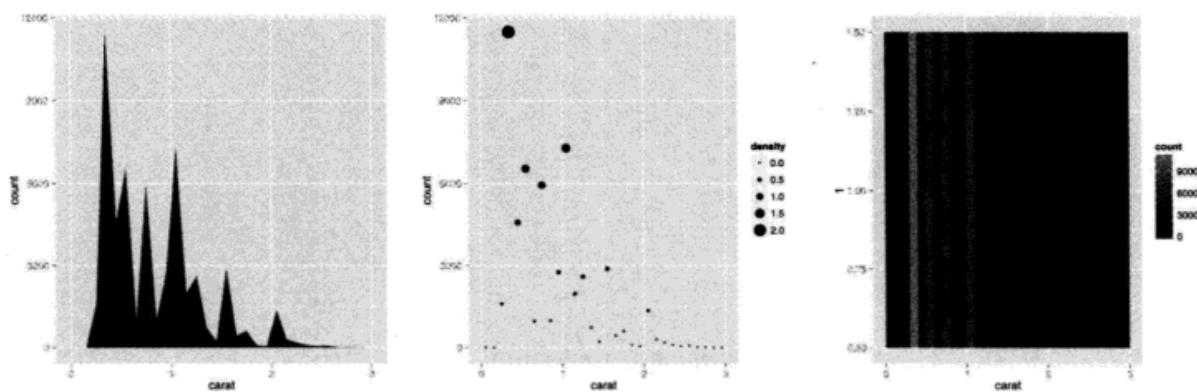


图 4.10 直方图的三种变体。频率多边形 (frequency polygon)(左); 散点图, 点的大小和高度都映射给了频率 (中); 热图 (heatmap) 用颜色来表示频率。

类似地, `ggplot2` 中有大量的几何对象是基于其他几何对象衍生出来的, 即修改某个已存在的几何对象的默认图形属性或统计变换。例如, 扰动几何对象 (`jitter geom`) 就是将点的位置调整设置为随机打乱。当某种图形变体将被多次使用, 或者在某种非常特殊的环境下使用时, 你便可以创建一个新的几何对象。表 4.6 列出了一些几何对象的“别名”。

表 4.6 通过修改其他几何对象默认值得到的几何对象。

别称	基础几何对象	默认值的更改
area	ribbon	<code>aes(min = 0, max = y), position = "stack"</code>
density	area	<code>stat = "density"</code>
freqpoly	line	<code>stat = "bin"</code>
histogram	bar	<code>stat = "bin"</code>
jitter	point	<code>position = "jitter"</code>
quantile	line	<code>stat = "quantile"</code>
smooth	ribbon	<code>stat = "smooth"</code>

4.9.2 显示已计算过的统计量

如果你有已经汇总过的数据，并且想直接用它，而不进行其他的统计变换，可以使用 `stat_identity()`，然后将合适的变量映射到相应的图形属性中。

4.9.3 改变图形属性和数据集

`ggplot2` 还有一个更强大的功能——将不同的数据画在不同的图层上。初看可能很奇怪：为什么要把不同的数据画到同一个图上呢？在实际应用中，你会经常遇到需要将相关数据集画到同一幅图上的情况。一个常见的例子即用拟合模型得出的预测值来扩充原数据集。虽然光滑曲线几何对象 (`smooth geom`) 可以向图中添加不同类型的光滑曲线，但是它不能替代外置的定量模型，无法总结出你对数据的理解。

我们再次回顾一下 4.5.3 节里用到过的 `Oxboys` 数据集。在图 4.4 里，我们给出了每个男孩的线性拟合曲线（简称之个体模型（左））和所有男孩的线性拟合曲线（简称之群体模型（右））。然而这两个模型都不完美：群体模型忽视了个体内的相关性；个体模型没能利用一般增长模型的信息，从而不能够精确地预测个体。实际上我们可以用混合模型来得到更好的结果。本节我们将探讨如何结合该模型输出的结果和原始数据来更深刻地理解数据和模型。

首先我们读入 `nlme` 包，然后拟合一个截距和斜率都包含随机效应的混合模型。（通过对个体模型的探索得知这个模型是合理的。）我们仍然要先建一个图形对象作为模板，这一步和图 4.3 的左图做法一致，然后我们利用从模型中导出的数据对模型进行渲染。

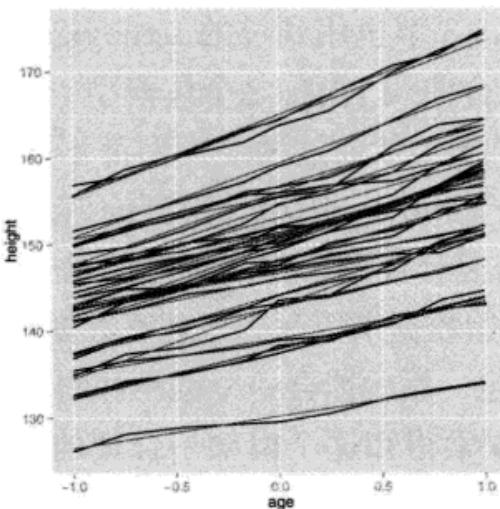
```
1 > require(nlme, quiet = TRUE, warn.conflicts = FALSE)
2 > model <- lme(height ~ age, data = Oxboys,
3 +   random = ~ 1 + age | Subject)
4 > oplot <- ggplot(Oxboys, aes(age, height, group = Subject)) +
5 +   geom_line()
```

随后，我们对预测的生长轨迹和实际的生长轨迹进行对比。我们先建立一个包含所有年龄 (`age`) 和个体 (`subjects`) 组合的网格数据框。对于这个简单的线性模型这么做虽然显得有点小题大做，但是仍作为例子放到这里，为以后处理更加复杂的模型提供参考。接下来我们把模型的预测值添加到刚刚生成的数据集中，变量名叫 `height`。

```
1 > age_grid <- seq(-1, 1, length = 10)
2 > subjects <- unique(Oxboys$Subject)
3
4 > preds <- expand.grid(age = age_grid, Subject = subjects)
5 > preds$height <- predict(model, preds)
```

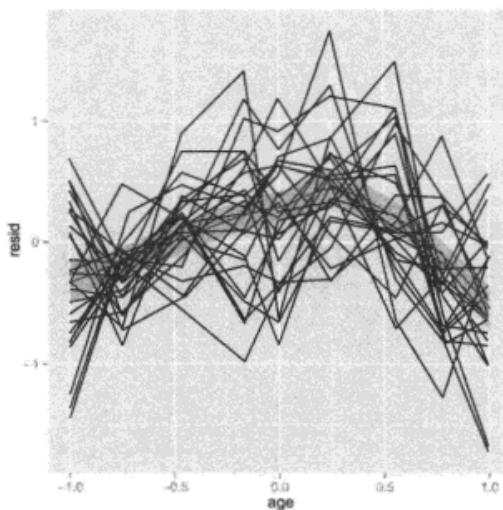
得到预测值后，我们把它和原始数据绘制到同一张图上。因为在新数据集 `preds` 里，我们使用了与原始数据 `Oxboys` 相同的变量名，并且我们想使用相同的分组图形属性，所以我们不用修改任何图形属性，只需要修改默认的数据集即可。我们还设定了颜色和大小两个图形属性参数以便于图形的比较。

```
1 > oplot + geom_line(data = preds, colour = "#3366FF", size= 0.4)
```



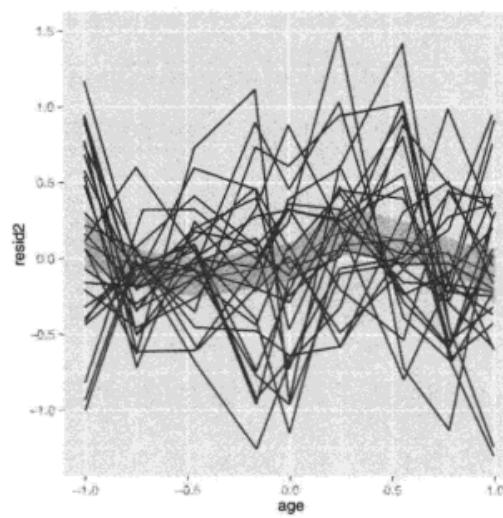
从图形上看，这个模型似乎很好地拟合了该数据更深层次的结构，但是仍然很难分辨细节。另一种比较模型拟合好坏的方法是观察残差，所以我们也演示一下这种方法。首先我们把拟合值 (`fitted`) 和残差 (`resid`) 都添加到原数据里去，然后更新数据集 (用 `%+%`)，将默认的 `y` 图形属性改成 `resid`，最后对整个数据添加一条光滑曲线。

```
1 > Oxboys$fitted <- predict(model)
2 > Oxboys$resid <- with(Oxboys, fitted - height)
3 >
4 > oplot %+% Oxboys + aes(y = resid) + geom_smooth(aes(group=1))
```



从图形可以看到残差并不是随机分布的，因此所建立的模型有缺陷。我们向模型中添加一个二次项，再次计算拟合值和残差并重新绘制残差图。这次没有明显的证据表明模型拟合得不好。

```
1 > model2 <- update(model, height ~ age + I(age ^ 2))
2 > Oxbboys$fitted2 <- predict(model2)
3 > Oxbboys$resid2 <- with(Oxbboys, fitted2 - height)
4 >
5 > oplot %+% Oxbboys + aes(y = resid2) + geom_smooth(aes(group=1))
```



值得注意的是，我们对图形对象的修改是非常容易的。我们更新了数据并且重新作了两次图却没有再次运行过 `oplot`，这正是 ggplot2 图层功能所秉承的理念：使得反复拟合和评估模型变得轻松而自然。

第 5 章 工具箱

5.1 简介

`ggplot2` 的图层化架构鼓励我们以一种结构化的方式来设计和构建图形。经过前几章的学习，你已经了解了图层的含义以及在图形上添加图层的方法，但可能尚不清楚有哪些几何对象和统计变换能够用来辅助构建表现力更强的图形。本章列举了 `ggplot2` 中大量几何对象和统计变换的一部分，并把它们按照用途进行了细分。本章旨在概述这些可用的几何对象和统计变换，故在文中并未逐个详述。如果你想逐个了解 `ggplot2` 中的几何对象以及它们的用法示例，请参阅在线文档和相应的程序帮助。本章所使用数据集的详细信息亦可在以上文档中找到。

本章的每一节都将解决一个特定的作图问题。这种分类方式并不是详尽无遗或独一无二的，即除此之外仍有许多其他可能的图形分类方式。只要你善于创造性地运用作图命令，每一种几何对象都可以有不同的用途。不过，以下的章节编排应该已经覆盖了许多常见任务，并能够帮助你学习各种可能的用法。本章分为以下几节：

- 基本的图形类型，详见 § 5.3，绘制常见的、“有名有姓”的图形，如散点图和折线图；
- 展示分布，详见 § 5.4，包括连续型分布和离散型分布、一维分布和二维分布、联合分布和条件分布；
- 应对散点图中的遮盖绘制问题，详见 § 5.5，这是大型数据集带来的一项挑战；
- 绘制曲面图，详见 § 5.6，即在二维平面上展示三维曲面；
- 统计汇总，详见 § 5.9，展示信息丰富的数据摘要；

- 绘制地图，详见 § 5.7；
- 揭示数据中的不确定性和误差，详见 § 5.8，用到多种一维和二维区间；
- 为图形添加注解，详见 § 5.10，使用补充信息来标注、描述并解释图形；
- 绘制加权的数据，详见 § 5.11。

本节中的示例将混合使用 `ggplot()` 和 `qplot()`，这更接近于实际的用法。如果你需要关于如何在这两者间转化的提示，请参阅附录 A.2。这些示例并不艰深，我们希望读者在阅读完本章后即可绘制出想要的图形。

5.2 图层叠加的总体策略

在添加图层之前弄清楚它的作用是十分必要的。总体来说，图层有三种用途：

- 用以展示数据本身。我们绘制原始数据的目的可能有很多，这依赖于我们辨识数据的整体结构、局部结构以及离群点等模式的技巧。本层几乎在每幅图形上都会出现。在探索数据的初始阶段，本层通常也是唯一的图层。
- 用以展示数据的统计摘要。随着数据探索和建模的深入，在数据的背景下来展示模型的预测效果是很有用的。我们可以从数据摘要中进一步理解数据，同时对模型作出评价。展示数据本身可以帮助我们改善模型，而展示模型可以帮助我们揭示数据的微妙之处，这也是我们可能漏掉的信息。本层通常绘制在数据层之上。

如果回顾前述章节中的示例，可能会发现许多示例图形的数据层上都叠加了一些图层，用以展示某种统计摘要。

- 用以添加额外的元数据 (**metadata**)、上下文信息和注解。元数据层展示了背景的上下文信息，也可为原始数据赋予有现实意义的注解。元数据作为图形背景和图形前景时同样有用。地图就经常作为空间数据的背景层。在绘制作为背景的元数据时，它不应该影响到主数据的展示，因此它往往被放置在主数据的下层，且在配色上尽可能不要太突出。也就是说，如果我们专注地看背景，我们可以很轻松地看到它，但它并不会在我们随意浏览图形时突兀地浮现。

其他的元数据可以用来强调数据中的重要特征 (feature)。例如，你也许希

望为一些拐点或离群点加上解释性的标签，并希望它们尽可能地突出显示。这种情况下，我们希望这种图层是最后绘制的图层。

5.3 基本图形类型

以下几何对象是 `ggplot2` 图形的基本组成部分。每种几何对象自身即可独立构建图形，同时也可以组合起来构建更复杂的几何对象。这些几何对象基本上都关联了一种常见的图形：当某幅图形只使用了一种几何对象构建时，这幅图往往拥有一个特定的名称。

这些几何对象均是二维的，故 `x` 和 `y` 两种图形属性都是不可或缺的。同时，它们都可以接受 `colour` 和 `size` 图形属性，另外，填充型几何对象（条形、瓦片（tile）和多边形）还可以接受 `fill` 图形属性。点使用 `shape` 图形属性，线和路径接受 `linetype` 图形属性。这些几何对象可用于展示原始数据，另行计算得到的数据摘要和元数据。

- `geom_area()` 用于绘制面积图（area plot），即在普通线图的基础上，依 `y` 轴方向填充了下方面积的图形。对于分组数据，各组将按照依次堆积的方式绘制。
- `geom_bar(stat = "identity")` 绘制条形图。我们需要指定 `stat = "identity"`，因为默认的统计变换将自动对“值”进行计数（所以本质上这是一个一维的几何对象，参见 § 5.4）。而统计变换 `identity` 将保持数据不变。

默认情况下，相同位置的多个条形图将以依次向上堆积的形式绘制。

- `geom_line()` 绘制线条图。图形属性 `group` 决定了哪些观测是连接在一起的；参考 4.5.3 节以了解更多细节。`geom_path` 与 `geom_line` 类似，但 `geom_path` 中的线条是根据它们在数据中出现的顺序进行连接的，而非从左至右进行连接。
- `geom_point()` 绘制散点图。
- `geom_polygon()` 绘制多边形，即填充后的路径。数据中的每一行代表了多边形的一个顶点。在绘图之前将多边形的顶点坐标数据和原始数据进行合并往往会更方便。5.7 节以地图数据为例详细地阐明了这个概念。
- `geom_text()` 可在指定点处添加标签。它是这些几何对象中唯一一个需要

额外图形属性的：它需要指定 `label` 参数。我们可以通过设置可选的图形属性 `hjust` 和 `vjust` 来控制文本的横纵位置；此外，可以设置图形属性 `angle` 来控制文本的旋转。你可以参考附录 B 以了解更多。

- `geom_tile()` 用来绘制色深图 (image plot) 或水平图 (level plot)。所有的瓦片 (tile) 构成了对平面的一个规则切分，且往往将 `fill` 图形属性映射至另一个变量。

使用以下代码绘制以上几何对象，结果如图 5.1 所示。

```
1 df <- data.frame(  
2   x = c(3, 1, 5),  
3   y = c(2, 4, 6),  
4   label = c("a", "b", "c")  
5 )  
6 p <- ggplot(df, aes(x, y)) + xlab(NULL) + ylab(NULL)  
7 p + geom_point() + labs(title = "geom_point")  
8 p + geom_bar(stat="identity") +  
9   labs(title = "geom_bar(stat=\"identity\")")  
10 p + geom_line() + labs(title = "geom_line")  
11 p + geom_area() + labs(title = "geom_area")  
12 p + geom_path() + labs(title = "geom_path")  
13 p + geom_text(aes(label = label)) + labs(title = "geom_text")  
14 p + geom_tile() + labs(title = "geom_tile")  
15 p + geom_polygon() + labs(title = "geom_polygon")
```

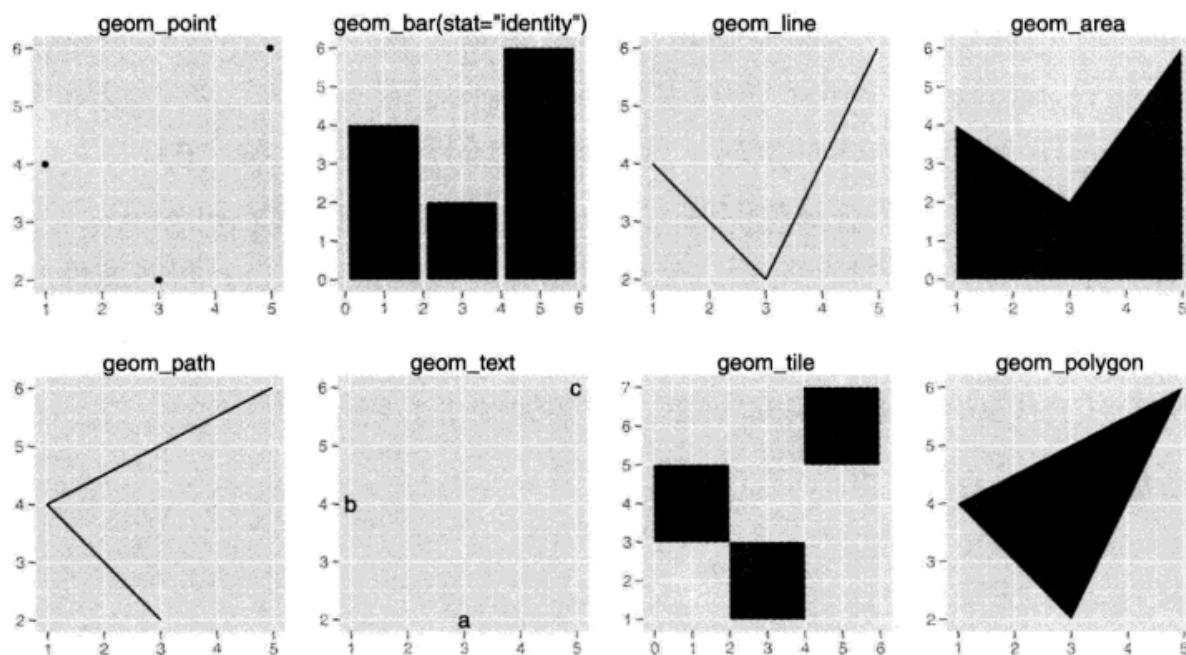


图 5.1 使用不同的基本几何对象绘制相同数据的效果。从左上到右下的图形名称分别为：散点图、条形图、线条图、面积图、路径图、含标签的散点图、色深图/水平图和多边形图。注意观察条形图、面积图和瓦片图的坐标轴范围：这三种几何对象占据了数据本身范围以外的空间，于是坐标轴被自动拉伸了。

5.4 展示数据分布

有一些几何对象可以用于展示数据的分布，具体使用哪种取决于分布的维度、分布是连续型或是离散型，以及我们感兴趣的是条件分布还是联合分布。

对于一维连续型分布，最重要的几何对象是直方图。图 5.2 使用直方图展示了 diamonds 数据中的 depth 变量。为了找到一个表现力强的视图，多次测试组距的布局细节是必不可少的。例如，我们可以改变组距宽度 (binwidth) 或者显式地精确指定切分位置 (breaks)。

有多种方式可以用来进行分布的跨组比较：同时绘制多个小的直方图，`facets = . ~ var`；使用频率多边形 (frequency polygon)，`geom = "freqpoly"`；或者使用条件密度图，`position = "fill"`。以下代码演示了这些方式，效果参见图 5.3。

```

1 depth_dist <- ggplot(diamonds, aes(depth)) + xlim(58, 68)
2 depth_dist +
3   geom_histogram(aes(y = ..density..), binwidth = 0.1) +
4   facet_grid(cut ~ .)
5 depth_dist + geom_histogram(aes(fill = cut), binwidth = 0.1,

```

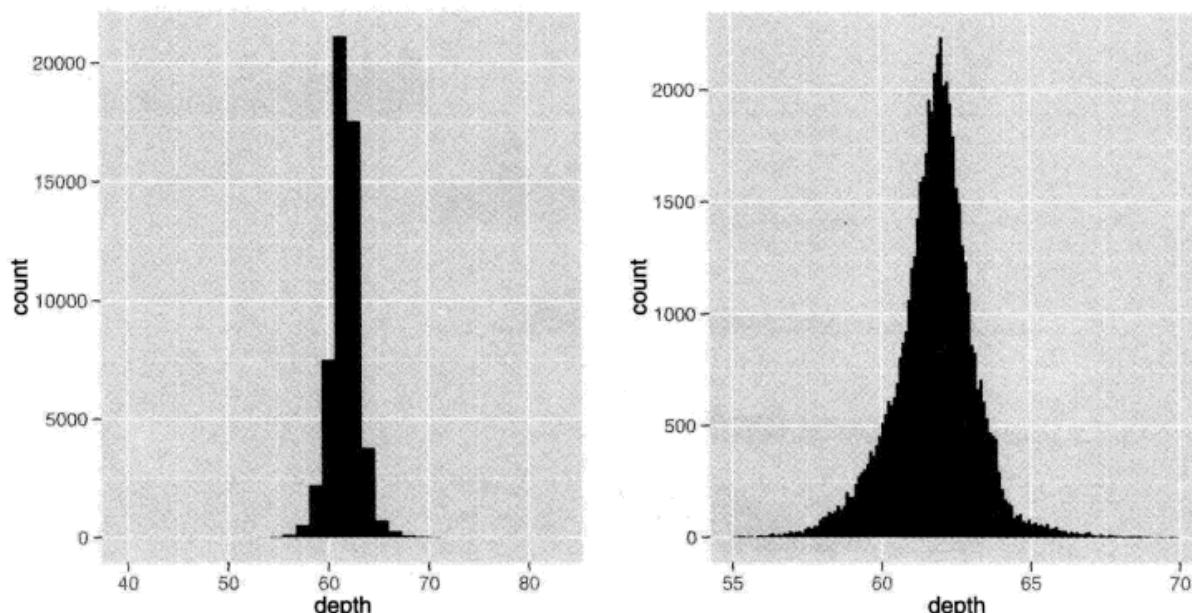


图 5.2 永远不要指望依靠默认的参数就能对某个具体的分布获得一个表现力强的图形(左图)。(右图)对 x 轴进行了放大, `xlim = c(55, 70)`, 并选取了一个更小的组距宽度, `binwidth = 0.1`, 较左图揭示出了更多细节。我们可以发现这个分布是轻度右偏的。同时别忘了在标题中写上重要参数(如组距宽度)的信息。

```

6   position = "fill")
7   depth_dist + geom_freqpoly(aes(y = ..density.., colour = cut),
8     binwidth = 0.1)

```

作为几何对象的直方图和频率多边形均使用了 `stat_bin` 统计变换。此统计变换生成了两个输出变量 `count` 和 `density`。变量 `count` 为默认值, 因为它的可解释性更好。而变量 `density` 基本上相当于 `count` 除以 `count` 的总数, 此变量在我们想要比较不同分布的形状而不是数据的绝对大小时更有用。特别地, 我们经常使用此变量来比较数据中不同大小子集的分布。

和分布相关的许多几何对象都是以几何对象 (geom)/统计变换 (stat) 的形式成对出现的。这些几何对象中大多数的本质都是别名 (alias): 一个基本几何对象结合一个统计变换, 即可绘制出想要的图形。表面上看, 箱线图 (boxplot) 似乎是一个例外, 但在幕后实现上, `geom_boxplot` 同样是使用基本的条、线和点组合而成的。

- `geom_boxplot = stat_boxplot + geom_boxplot`: 箱线图, 即一个连续型变量针对一个类别型变量取条件所得的图形。当类别型变量有许多独立的取值时, 这种图形比较有用。不过当类别型变量的取值很少时, 仿照上文直接研究分布的具体形状更佳。箱线图也可对连续型变量取条件, 前提

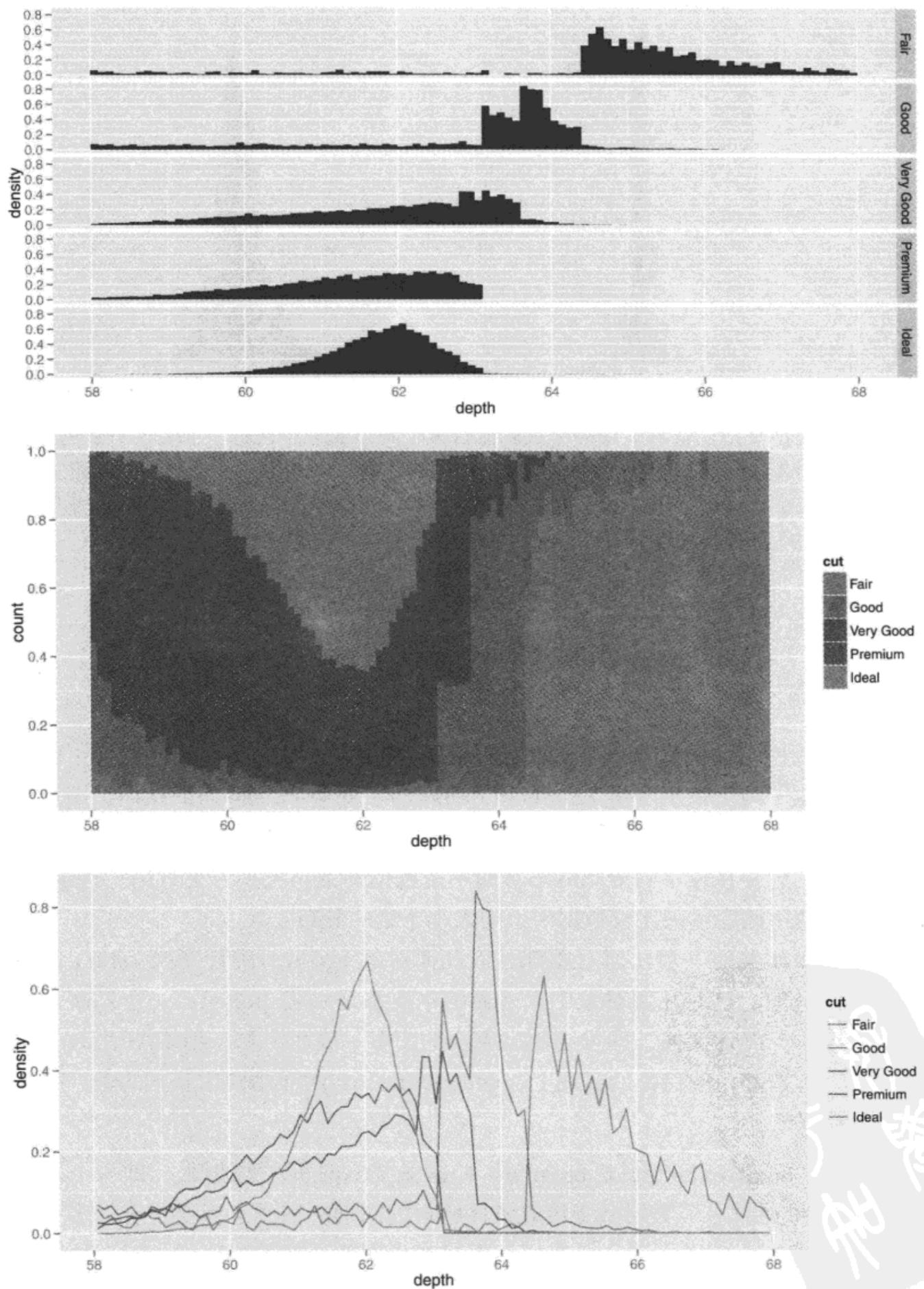


图 5.3 (参见彩图 14) 钻石数据切割和深度分布的三种视图。从上至下分别是分面直方图、条件密度图和频率多边形图。它们都显示了一个有趣的模式：随着钻石质量的提高，分布逐渐向左偏移且愈发对称。

是数据预先经过巧妙地封箱 (binning) 处理。图 5.4 展示了针对类别型或连续型变量取条件所得的箱线图^①。

```
1 library(plyr)
2 qplot(cut, depth, data = diamonds, geom = "boxplot")
3 qplot(carat, depth, data = diamonds, geom = "boxplot",
4   group = round_any(carat, 0.1, floor), xlim = c(0, 3))
```

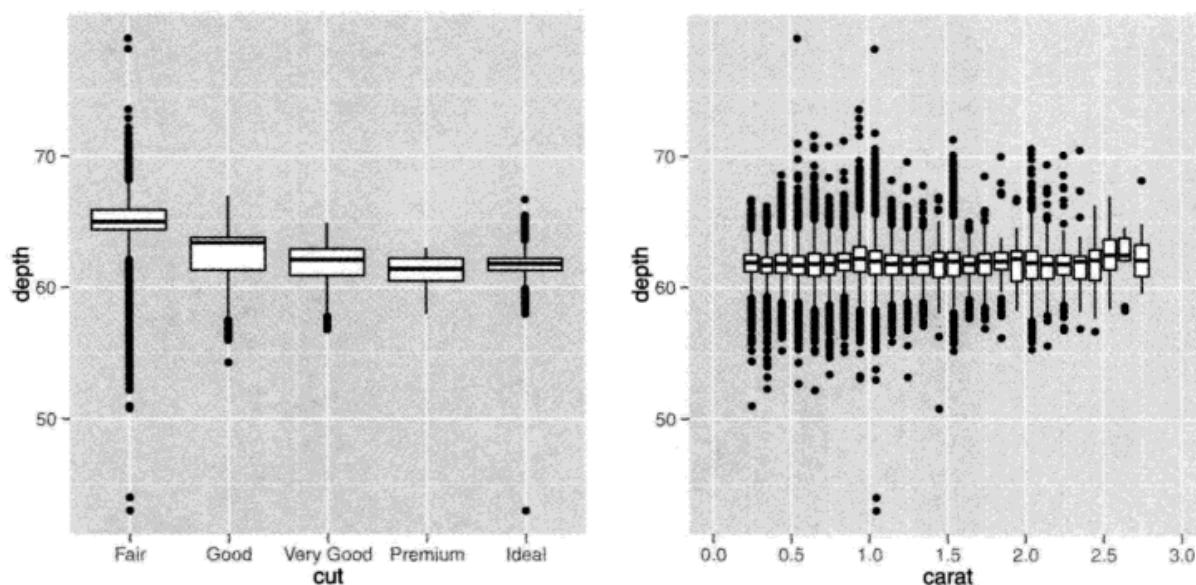


图 5.4 箱线图可以用于观察针对一个类别型变量 (如 `cut`) 取条件时 (左图), 或连续型变量 (如 `carat`) 取条件时 (右图), 连续型变量的分布情况。对于连续型变量, 必须设置 `group` 图形属性以得到多个箱线图。此处使用了 `group = round_any(carat, 0.1, floor)` 来获得针对变量 `carat` 以 0.1 个单位为大小封箱后的箱线图。

- `geom_jitter = position_jitter + geom_point`: 通过在离散型分布上添加随机噪声以避免遮盖绘制问题, 这是一种较为粗糙的方法。使用以下代码绘制的图 5.5 展示了这样的一个例子。

```
1 qplot(class, cty, data = mpg, geom = "jitter")
2 qplot(class, drv, data = mpg, geom = "jitter")
```

- `geom_density = stat_density + geom_area`: 基于核平滑方法进行平滑后得到的频率多边形, 如 2.5.3 节中所述。请仅在已知潜在的密度分布为平滑、连续且无界的时候使用这种密度图。可以使用参数 `adjust` 来调整所得密度曲线的平滑程度。使用以下代码绘制的图 5.6 展示了这样一个例子。

^①译者注: 图 5.4 原书代码中调用的函数 `round_any()` 在新版本中已被剥离并整合到 `plyr` 包中, 这里需要使用 `library(plyr)` 载入此包。

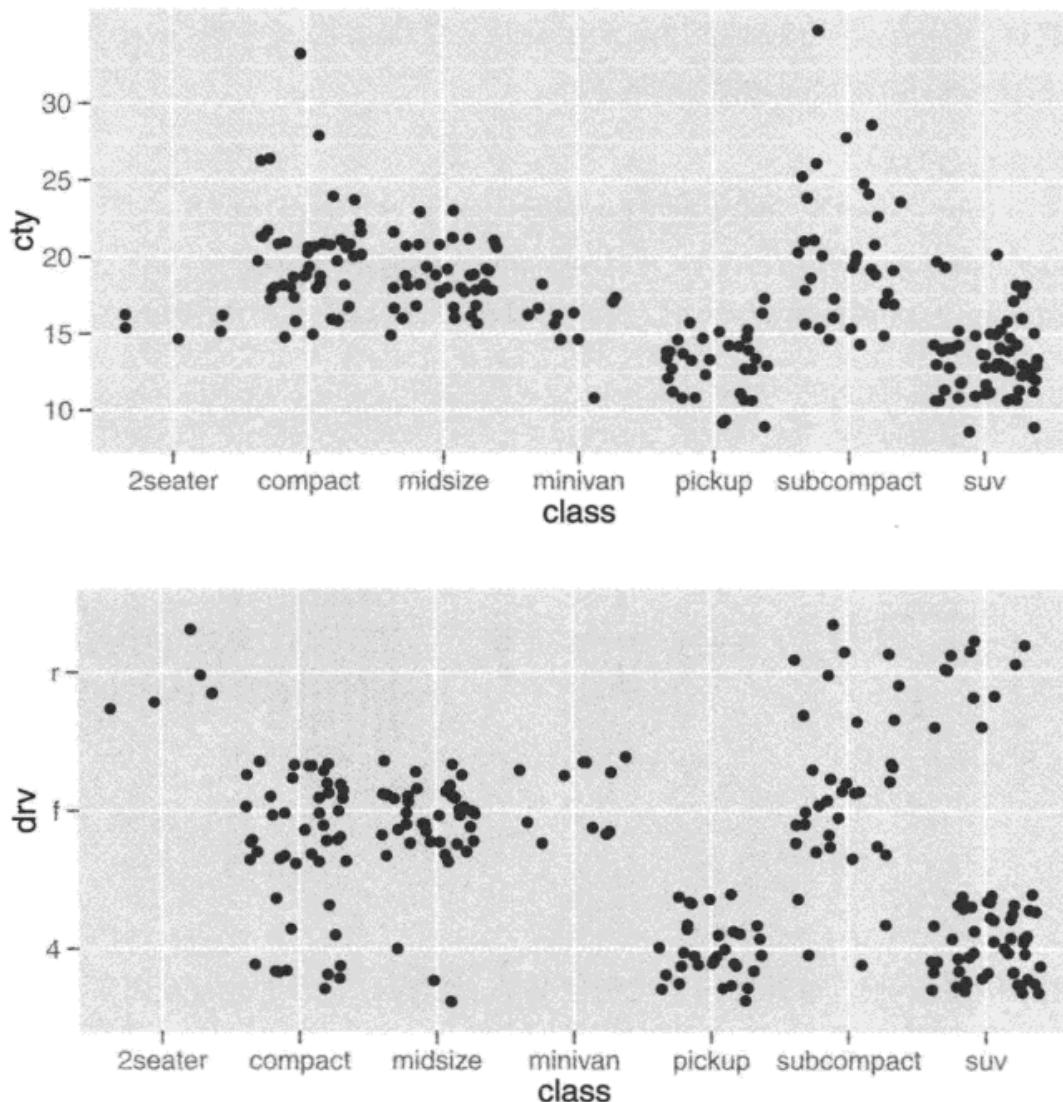


图 5.5 几何对象 jitter 可在二维分布中有一个离散型变量时绘制出一个较为粗略的图形。总体来说，这种数据打散的处理对小数据集更有效。上图展示了 mpg 数据集中离散型变量 class 和连续型变量 cty，下图则将连续型变量 cty 替换为离散型变量 drv。

```
1 qplot(depth, data = diamonds, geom = "density", xlim = c(54, 70))  
2 qplot(depth, data = diamonds, geom = "density", xlim = c(54, 70),  
3   fill = cut, alpha = I(0.2))
```

下节将讨论二维连续型联合分布的可视化。

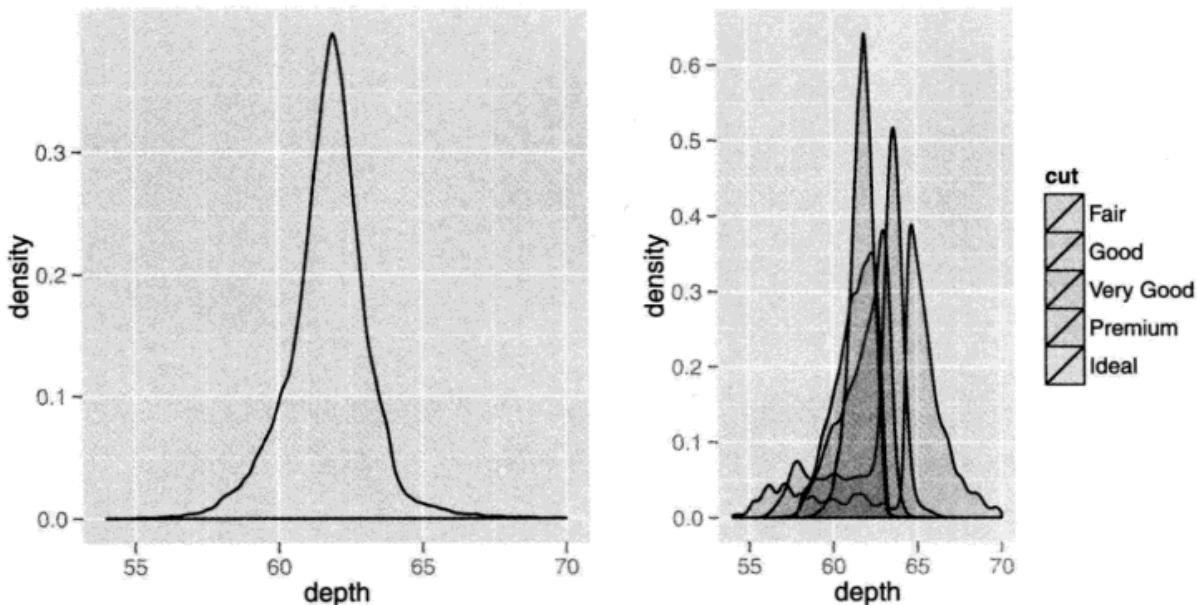


图 5.6 (参见彩图 15) 密度图实际上就是直方图的平滑化版本。它的理论性质比较理想，但难以由图回溯到数据本身。左图为变量 `depth` 的密度图。右图为按照变量 `cut` 的不同取值上色的版本。

5.5 处理遮盖绘制问题

散点图是研究两个连续型变量间关系的重要工具。但是当数据量很大时，这些点经常会出现重叠现象，从而掩盖真实的关系。在极端情况下，我们甚至只能看到数据所在的大致范围，根据这种图形作出的任何结论都是值得怀疑的。这种问题被称为遮盖绘制 (overplotting)，对付它有许多方法：

- 小规模的遮盖绘制问题可以通过绘制更小的点加以缓解，或者使用中空的符号，如图 5.7 所示。所用数据是从两个独立的正态分布中抽样所得的 2000 个点，所用代码如下：

```

1 df <- data.frame(x = rnorm(2000), y = rnorm(2000))
2 norm <- ggplot(df, aes(x, y))
3 norm + geom_point()
4 norm + geom_point(shape = 1)
5 norm + geom_point(shape = ".") ## 点的大小为像素级

```

- 对于更大的数据集产生的更为严重的遮盖绘制问题，我们可以使用 α 混合 (调整透明度) 让点呈现透明效果。假设我们以比值形式指定 α 的值，则分母代表的是一个位置的颜色变为完全不透明时所需重叠点的数量。在

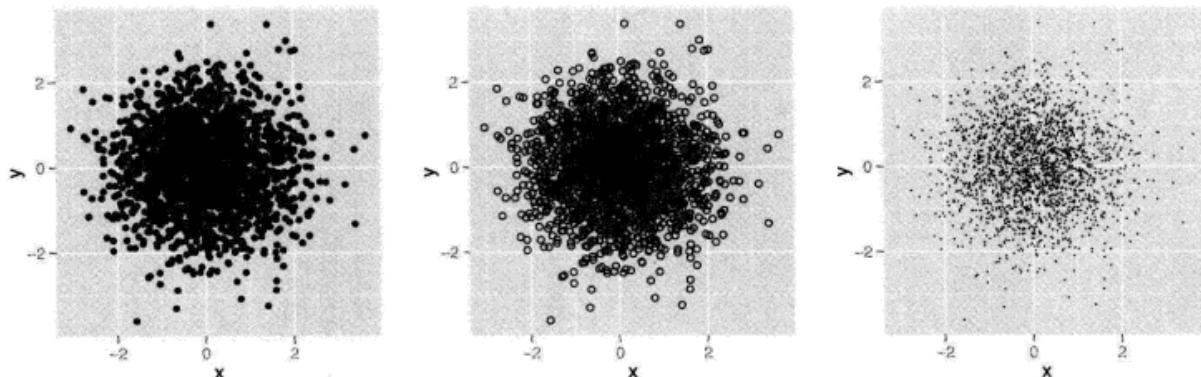


图 5.7 修改使用的符号可以帮助我们处理轻微到中等程度的遮盖绘制问题。从左至右分别为：默认的 `shape`、`shape = 1`(中空的点)，以及 `shape= ". "`(像素大小的点)。

R 中，可用的最小透明度为 $1/256$ ，所以对于严重的遮盖绘制问题，这种方法的效果并不会太好。图 5.8 演示了一些取值的效果，代码如下^②。

```
1 norm + geom_point(colour = "black", alpha = 1/3)
2 norm + geom_point(colour = "black", alpha = 1/5)
3 norm + geom_point(colour = "black", alpha = 1/10)
```

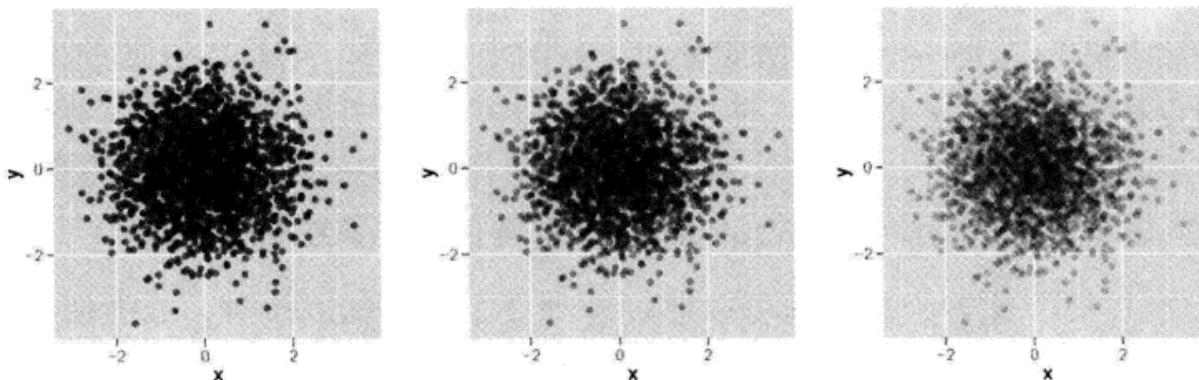


图 5.8 以从一个二元正态数据中抽样所得的数据为例，使用 α 混合来减轻遮盖绘制问题。 α 值从左至右分别为： $1/3, 1/5, 1/10$ 。

- 如果数据存在一定的离散性，我们可以通过在点上增加随机扰动来减轻重叠。特别是在与透明度一起使用时，这种方法很有效。默认情况下，增加的扰动量是数据分辨率 (resolution) 的 40%，这样可为数据中的邻接区域留下一定的小间隙。在图 5.9 中，diamond 数据中的变量 `table` 依最近的

^②译者注：与原书代码不同的是，这里直接使用了与参数 `colour` 独立的参数 `alpha` 来指定透明度。原书代码中的函数 `alpha()` 在新版本中已被剥离并整合到 `scales` 包中。关于标度 (scale) 的更多知识，将在下一章中介绍。

整数作取整处理，所以此处也可以设置两个连续整数间距的一半（即 0.5）作为打散的宽度。完整的代码如下^③。

```

1 td <- ggplot(diamonds, aes(table, depth)) +
2   xlim(50, 70) + ylim(50, 70)
3 td + geom_point()
4 td + geom_jitter()
5 jit <- position_jitter(width = 0.5)
6 td + geom_jitter(position = jit)
7 td + geom_jitter(position = jit, colour = "black", alpha = 1/10)
8 td + geom_jitter(position = jit, colour = "black", alpha = 1/50)
9 td + geom_jitter(position = jit, colour = "black", alpha = 1/200)

```

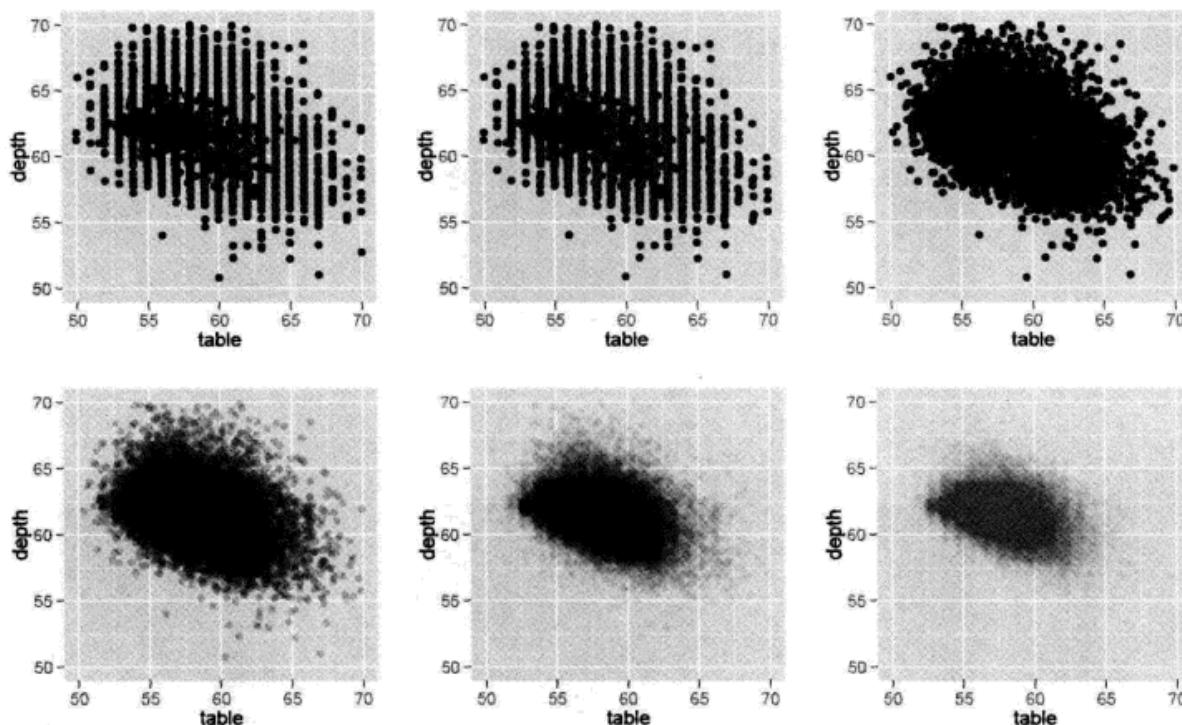


图 5.9 diamond 数据中的变量 `table` 和变量 `depth` 组成的图形，展示了如何使用数据打散和 α 混合来减轻离散型数据中的遮盖绘制问题。从左至右为：不加任何处理的点，使用默认扰动参数打散后的点，横向扰动参数为 0.5（横轴单位距离的一半）时打散后的点， α 取值 1/10， α 取值 1/50， α 取值 1/200。

受此启发，我们也可以认为遮盖绘制问题是一种二维核密度估计问题，于是又可引申出以下两种方法：

- 将点分箱并统计每个箱中点的数量，然后通过某种方式可视化这个数量

^③译者注：此图中对第二行三幅图原始绘图代码的修改与图 5.8 类似，均是使用与参数 `colour` 独立的参数 `alpha` 来指定透明度。

(直方图的二维推广)。将图形划分为小的正方形箱可能会产生分散注意力的视觉假象。Carr 等人 (1987) 建议使用六边形代之, 这类图形可以使用 `geom_hexagon` 这一几何对象实现, 它使用了 `hexbin` 包 (Carr et al., 2008) 提供的功能。图 5.10 对比了正方形箱和六边形箱的效果, 使用了参数 `bins` 和 `binwidth` 来控制箱的数量和大小。完整代码如下:

```

1 d <- ggplot(diamonds, aes(carat, price)) + xlim(1, 3) +
2   theme(legend.position = "none")
3 d + stat_bin2d()
4 d + stat_bin2d(bins = 10)
5 d + stat_bin2d(binwidth = c(0.02, 200))
6 d + stat_binhex()
7 d + stat_binhex(bins = 10)
8 d + stat_binhex(binwidth = c(0.02, 200))

```

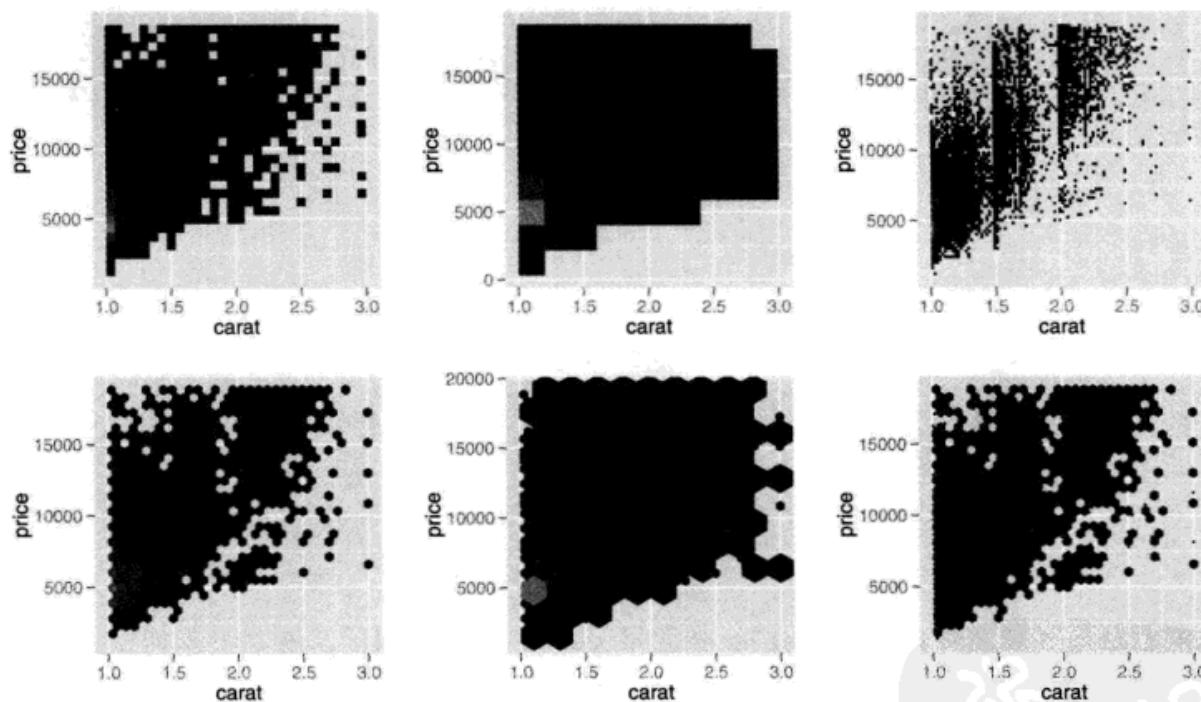


图 5.10 第一行使用正方形显示分箱, 第二行使用六边形显示。左栏使用默认分箱参数, 中栏使用参数 `bins = 10`, 右栏使用参数 `binwidth = c(0.02, 200)`。为了节约空间, 均略去了图例。

- 使用 `stat_density2d` 作二维密度估计, 并将等高线添加到散点图中, 或以着色瓦片 (colored tiles) 直接展示密度, 或使用大小与分布密度成比例的点进行展示。图 5.11 展示了部分选项的效果, 代码如下:

```

1 d <- ggplot(diamonds, aes(carat, price)) + xlim(1, 3) +

```

```
2   theme(legend.position = "none")
3 d + geom_point() + geom_density2d()
4 d + stat_density2d(geom = "point", aes(size = ..density..),
5   contour = F) + scale_size_area()
6 d + stat_density2d(geom = "tile", aes(fill = ..density..),
7   contour = F)
8 last_plot() + scale_fill_gradient(limits = c(1e-5,8e-4))
```

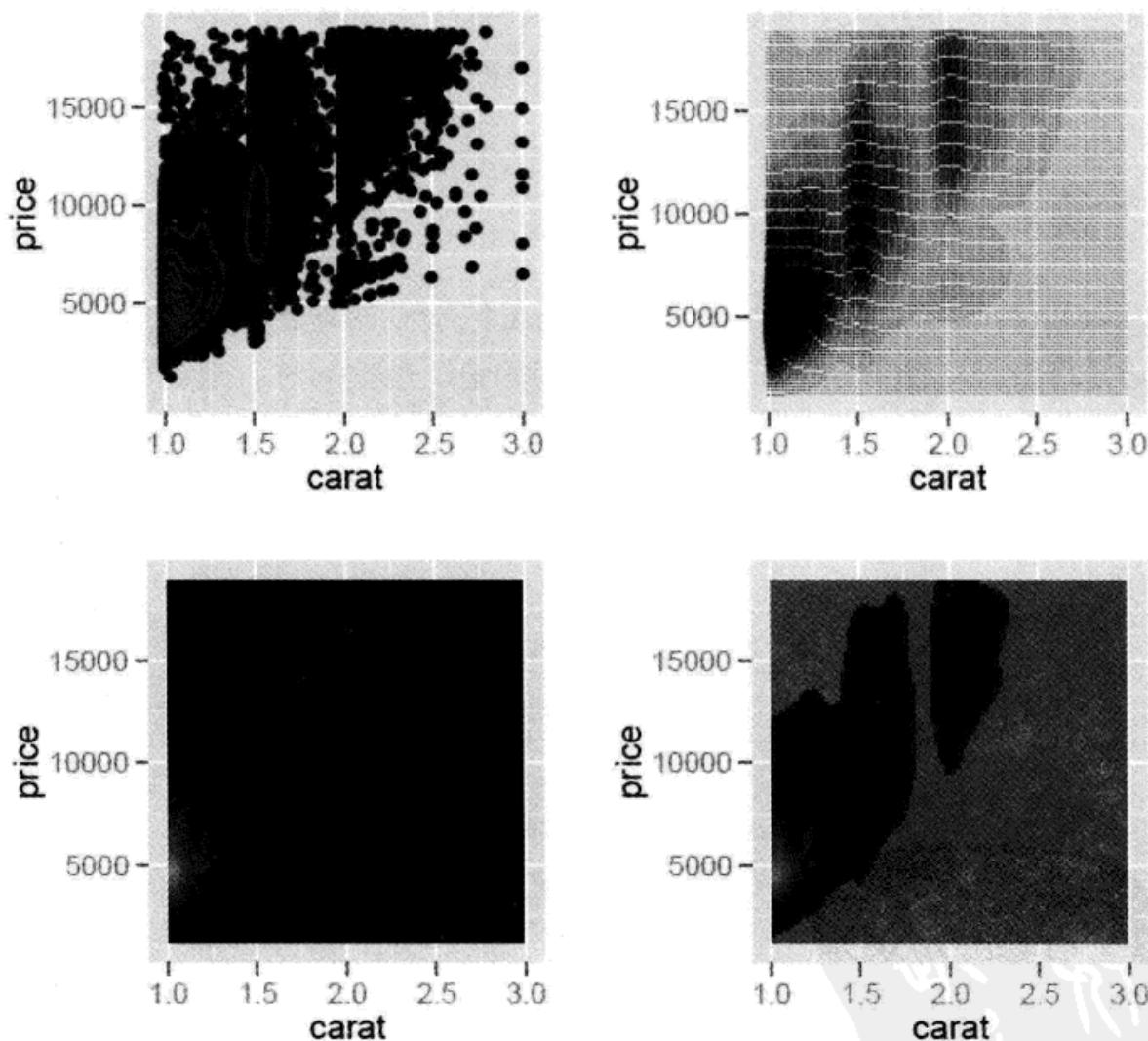


图 5.11 使用密度估计对点的密度建模并进行可视化。上图为基于点和等高线的密度展示，下图为基于色深的密度展示。

- 如果我们对给定 x 时 y 的条件分布感兴趣，那么 2.5.3 节中所述的技术也会是有用的。

对付遮盖绘制问题的另外一种方法是在图形上添加数据摘要，以指引人眼在茫茫数据中发现所寻模式的真实形状。例如，我们可以使用 `geom_smooth` 添加一条平滑曲线来展示数据的中心。本书 5.9 节中给出了更多的思路。

5.6 曲面图

`ggplot2` 暂不支持真正的三维曲面图。但具有在二维平面上展现三维曲面的常见工具：等高线图，着色瓦片（colored tiles）以及气泡图。这些图形在上节中被用来绘制二维密度分布面。对于三维交互式图形和真实三维曲面的绘制，建议关注一下 RGL，<http://rgl.neoscientists.org/about.shtml>。

5.7 绘制地图

`ggplot2` 提供了一些工具，让使用 `maps` 包绘制的地图与其他 `ggplot2` 图形的结合变得十分方便^④。表 5.1 列举了可用的地图数据名称，遗憾的是，这些地图主要是针对美国本土的^⑤。我们使用地图数据可能有两种主要原因：一是为空间数据图形添加参考轮廓线，二是通过在不同的区域填充颜色以构建等值线图（choropleth map）。

表 5.1 `maps` 包中可用的地图。

国家	地图数据名
法国	france
意大利	italy
新西兰	nz
美国（郡级）	county
美国（州级）	state
美国（边界）	usa
全世界	world

^④译者注：较新版本的 `ggplot2` 中引入的几何对象 `geom_map()`，可以大大简化等高线图的绘制过程。

^⑤译者注：关于中国地图的绘制，可参考《用 R 软件绘制中国分省市地图》一文。（<http://cos.name/2009/07/drawing-china-map-using-r/>）

添加地图边界可通过函数 `borders()` 来完成。函数的前两个参数指定了要绘制的地图名 `map` 以及其中的具体区域 `region`, 其余的参数用于控制边界的外观: 如边界的颜色 `colour` 和线条粗细 `size`。如果我们想要的是填充颜色的多边形而不是单纯的边界, 可以通过设定参数 `fill` 来实现。以下代码使用 `borders()` 展示了图 5.12 中的空间数据^⑥。

```

1 library(maps)
2 data(us.cities)
3 big_cities <- subset(us.cities, pop > 500000)
4 qplot(long, lat, data = big_cities) + borders("state", size = 0.5)
5 tx_cities <- subset(us.cities, country.etc == "TX")
6 ggplot(tx_cities, aes(long, lat)) +
7   borders("county", "texas", colour = "grey70") +
8   geom_point(colour = "black", alpha = 0.5)

```

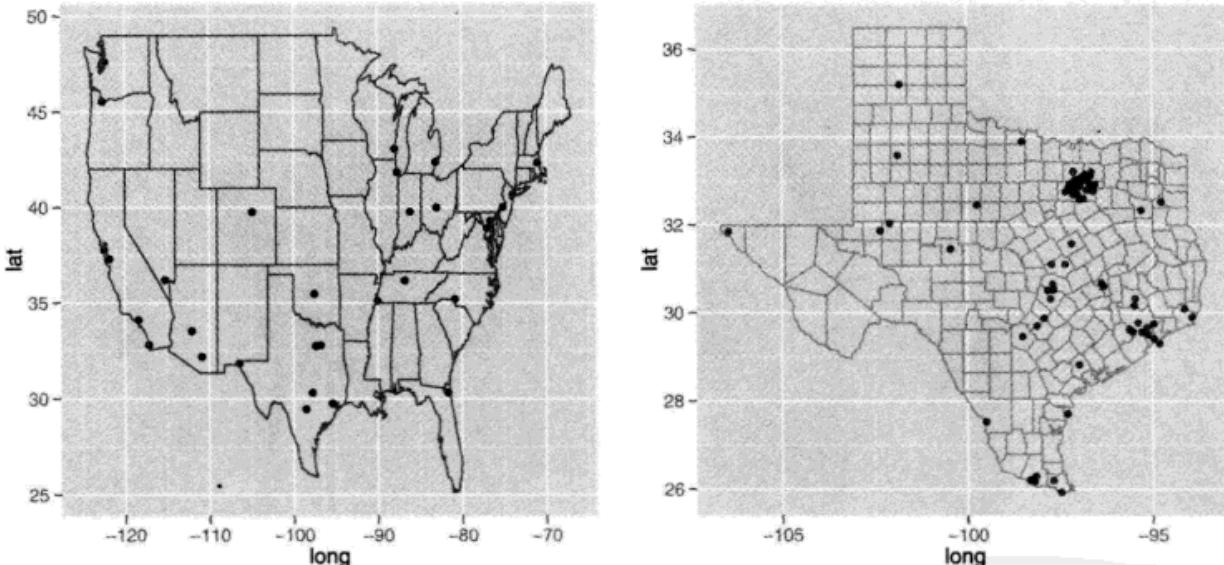


图 5.12 函数 `borders()` 的使用实例。左图展示了美国 (2006 年 1 月) 五十万人口以上的城市, 右图为德克萨斯州的城市区划。

等值线图 (choropleth map) 则相对更难处理一些, 自动化程度也没那么高, 原因在于, 要将我们数据中的标识符 (identifier) 同地图数据中的标识符完全匹配起来是有一定挑战性的。以下实例展示了如何使用 `map_data()` 将地图数据转换为数据框, 此数据框可以在之后通过 `merge()` 操作与我们的数据相融合, 最

^⑥译者注: 与原书代码不同的是, 这里的第二幅图直接使用了与参数 `colour` 独立的参数 `alpha` 来指定透明度。

终绘制出等值线图。结果如图 5.13 所示。我们数据中的细节可能不同，但关键在于，我们的数据和地图数据中要有一列可以相互匹配。

```

1 library(maps)
2 states <- map_data("state")
3 arrests <- USArrests
4 names(arrests) <- tolower(names(arrests))
5 arrests$region <- tolower(rownames(USArrests))
6
7 choro <- merge(states, arrests, by = "region")
8 ## 由于绘制多边形时涉及顺序问题
9 ## 且 merge 破坏了原始排序 故将行重新排序
10 choro <- choro[order(choro$order), ]
11 qplot(long, lat, data = choro, group = group,
12     fill = assault, geom = "polygon")
13 qplot(long, lat, data = choro, group = group,
14     fill = assault / murder, geom = "polygon")

```

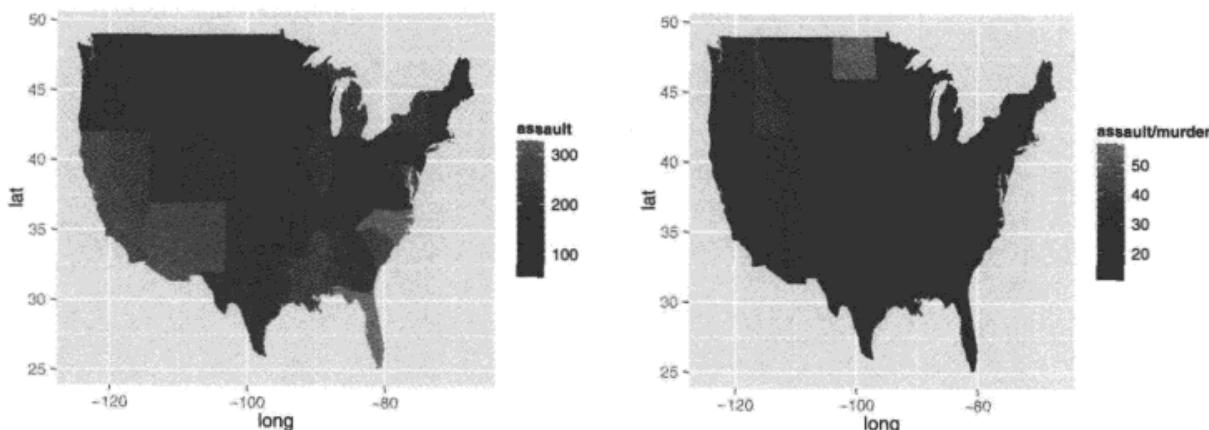


图 5.13 (参见彩图 16) 左侧的等值线图展示了各州人身伤害案件的数量，右侧的等值线图展示了人身伤害和谋杀类案件的比率。

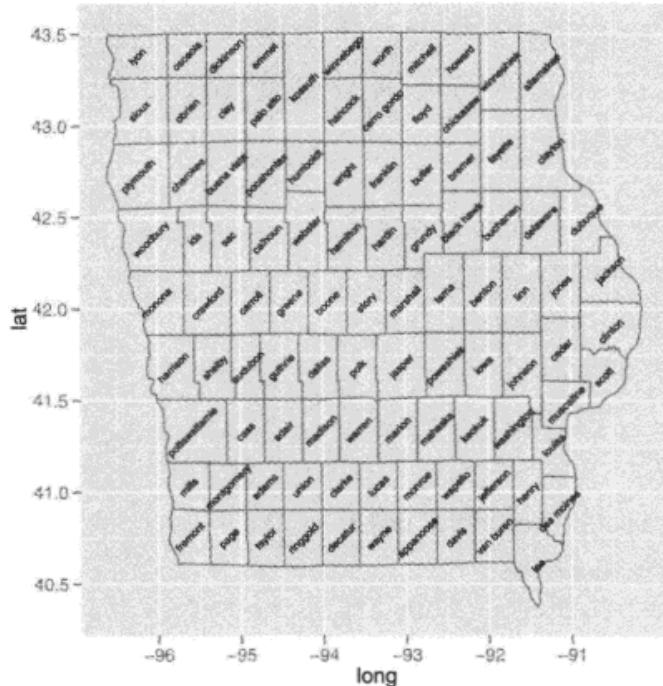
如果我们想对数据做进一步处理，函数 `map_data()` 也是很有用的。下例中，我们计算了爱荷华州每个郡的（近似）中心，然后利用这些中心位置数据在地图上对其名称进行标注。

```

1 > library(plyr) ## ddply() 在新版本中已被剥离并整合到 plyr 包中，这里先载入该包
2 > ia <- map_data("county", "iowa")
3 > mid_range <- function(x) mean(range(x, na.rm = TRUE))
4 > centres <- ddply(ia, .(subregion),

```

```
5 + colwise(mid_range, .(lat, long)))
6 > ggplot(ia, aes(long, lat)) +
7 + geom_polygon(aes(group = group),
8 + fill = NA, colour = "grey60") +
9 + geom_text(aes(label = subregion), data = centres,
10 + size = 2, angle = 45)
```



5.8 揭示不确定性

不论是从模型所得或是从对分布的假设而得，如果我们已经知道了一些关于数据中不确定性的信息，那么对这些信息加以展示通常是很重要的。在 `ggplot2` 中，共有四类几何对象可以用于这项工作，具体使用哪个取决于 `x` 的值是离散型还是连续型的，以及我们是否想要展示区间内的中间值，或是仅仅展示区间。这些几何对象列于表 5.2 中。它们均假设我们对给定 `x` 时 `y` 的条件分布感兴趣，并且都使用了图形属性 `ymin` 和 `ymax` 来确定 `y` 的值域。如果你想绘制出相反的结果，请参看 7.3.3 节中的 `coord_flip`。

由于标准误的计算方式有很多，所以具体如何计算将由你自己决定。对于比较简单的情况，`ggplot2` 提供了部分数据摘要计算函数（在 5.9 节中有阐述），不过，我们完全可以自行计算。对于线性模型，`effects` 包（Fox, 2008）非常适合提取其中的这类值。下例拟合了一个双因素含交互效应回归模型，并且展示了

表 5.2 用于展示区间的几何对象。它们有助于不确定性信息的可视化。

变量 X 类型	仅展示区间	同时展示区间和中间值
连续型	geom_ribbon	geom_smooth(stat = "identity")
离散型	geom_errorbar	geom_crossbar
	geom_linerange	geom_pointrange

如何提取边际效应 (marginal effects) 和条件效应 (conditional effects)，以及如何将其可视化。图 5.15 侧重于对类别型变量 `color` 的研究，而图 5.16 侧重于对连续型变量 `carat` 的分析。

```

1 > d <- subset(diamonds, carat < 2.5 &
2 + rbinom(nrow(diamonds), 1, 0.2) == 1)
3 > d$lcarat <- log10(d$carat)
4 > d$lprice <- log10(d$price)
5 >
6 > # 剔除整体的线性趋势
7 > detrend <- lm(lprice ~ lcarat, data = d)
8 > d$lprice2 <- resid(detrend)
9 >
10 > mod <- lm(lprice2 ~ lcarat * color, data = d)
11 >
12 > library(effects)
13 > effectdf <- function(...) {
14 + suppressWarnings(as.data.frame(effect(...)))
15 + }
16 > color <- effectdf("color", mod)
17 > both1 <- effectdf("lcarat:color", mod)
18 >
19 > carat <- effectdf("lcarat", mod, default.levels = 50)
20 > both2 <- effectdf("lcarat:color", mod, default.levels = 3)

1 ## 图 5.14:
2 qplot(lcarat, lprice, data=d, colour = color)
3 qplot(lcarat, lprice2, data=d, colour = color)
4 ## 图 5.15:
5 fplot <- ggplot(mapping = aes(y = fit, ymin = lower, ymax = upper)) +
6   ylim(range(both2$lower, both2$upper))

```

```

7 fplot %+% color + aes(x = color) + geom_point() + geom_errorbar()
8 fplot %+% both2 +
9   aes(x = color, colour = lcarat, group = interaction(color, lcarat)) +
10  geom_errorbar() + geom_line(aes(group=lcarat)) +
11  scale_colour_gradient()
12 ## 图 5.16:
13 fplot %+% carat + aes(x = lcarat) + geom_smooth(stat = "identity")
14
15 ends <- subset(both1, lcarat == max(lcarat))
16 fplot %+% both1 + aes(x = lcarat, colour = color) +
17   geom_smooth(stat="identity") +
18   scale_colour_hue() + theme(legend.position = "none") +
19   geom_text(aes(label = color, x = lcarat + 0.02), ends):

```

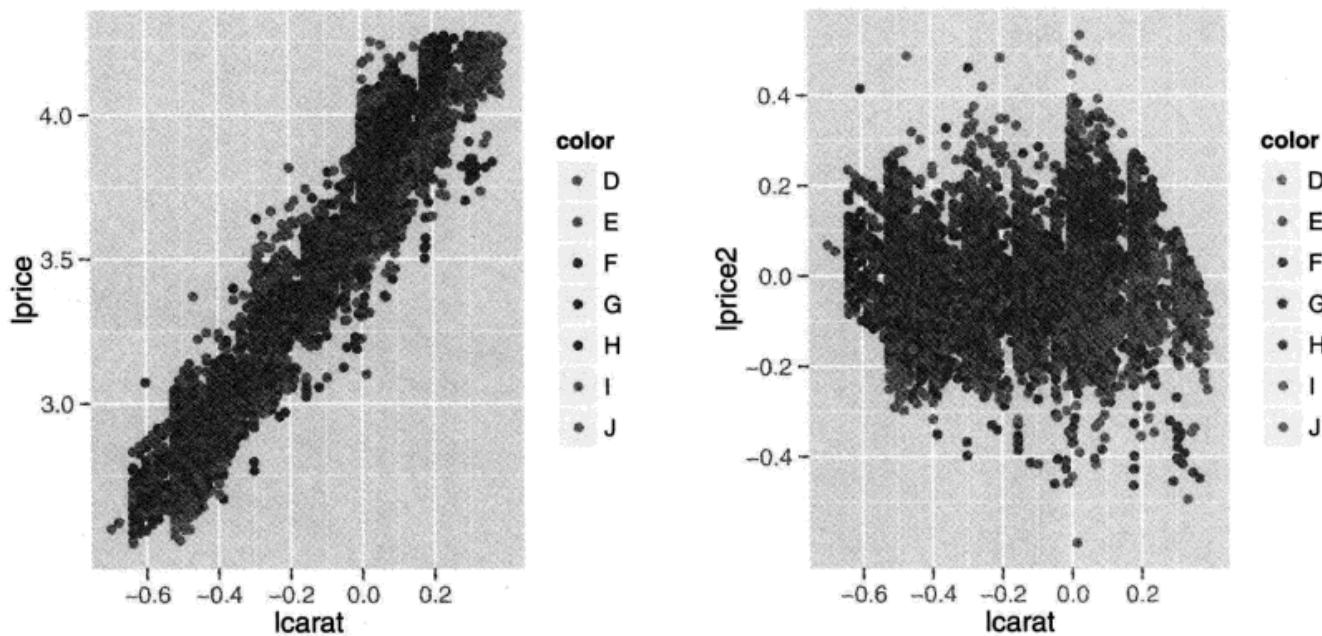


图 5.14 进行数据变换以移除显而易见的效应。左图对 x 轴和 y 轴的数据均取以 10 为底的对数以剔除非线性性。右图剔除了主要的线性趋势。

注意，在为这类图形添加题注时，我们需要细致地描述其中所含置信区间的本质，并说明观察置信区间之间的重叠是否有意义（说明：当比较不同组时，如果区间没有重叠，则说明差异显著）。即，这些标准误是针对单组的均值的，还是针对不同组件均值之差的。在计算和展示这些标准误时，`multcomp` 包和 `multcompView` 包将非常有用，同时他们在多重比较中能正确地对自由度进行调整。

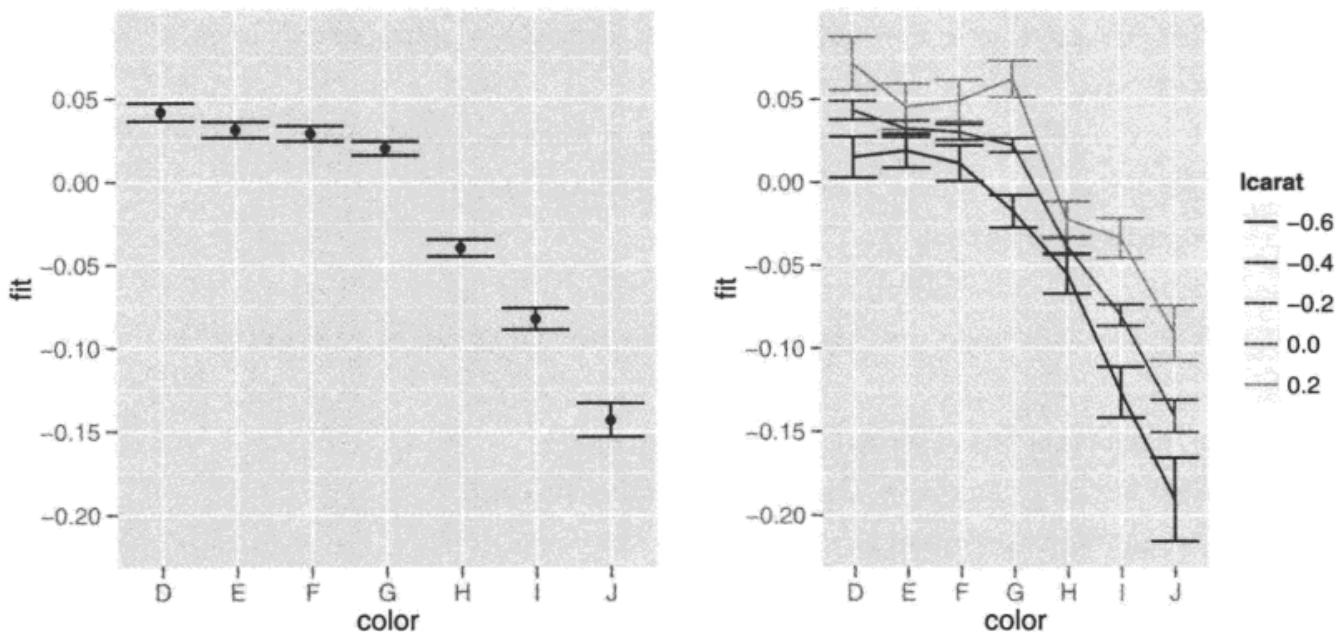


图 5.15 (参见彩图 17) 展示模型估计结果中变量 `color` 的不确定性。左图为 `color` 的边际效应。右图则是针对变量 `caret` 的不同水平 (level), 变量 `color` 的条件效应。误差棒显示了 95% 的逐点置信区间。

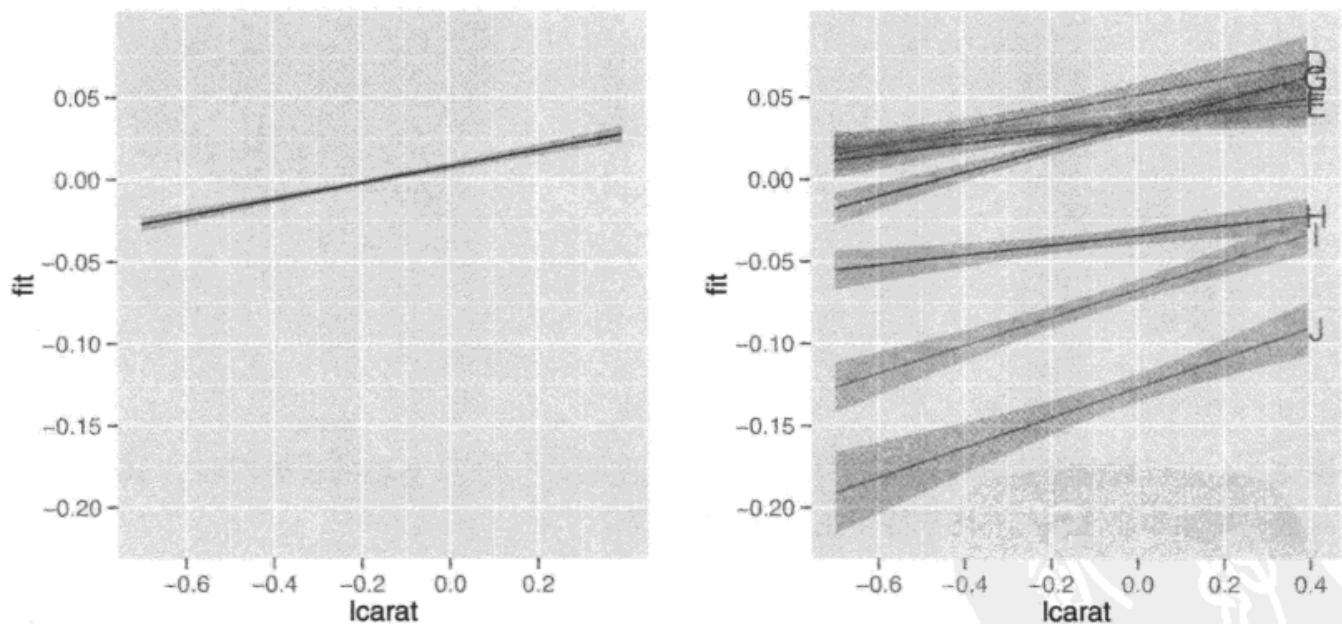


图 5.16 (参见彩图 18) 展示模型估计结果中变量 `carat` 的不确定性。左图为 `carat` 的边际效应。右图则是针对变量 `color` 的不同水平, 变量 `carat` 的条件效应。误差带显示了 95% 的逐点置信区间。

5.9 统计摘要

对于每个 x 的取值，计算对应 y 值的统计摘要通常是很常用的。在 `ggplot2` 中，这一角色由 `stat_summary()` 担当，它使用 `ymin`, `y` 和 `ymax` 等图形属性，为汇总 y 的条件分布提供了一种灵活的方式。图 5.17 展示了此工具可以计算的多种摘要类型中的一部分。

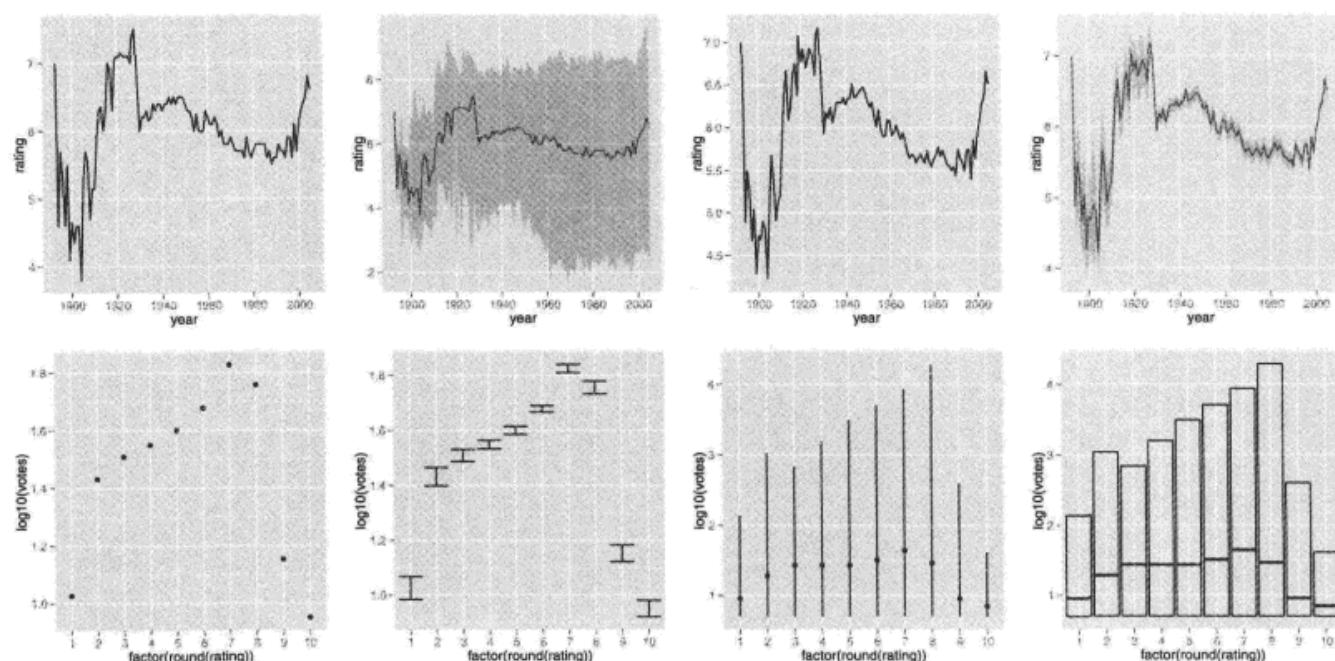


图 5.17 函数 `stat_summary` 的使用示例。首行从左至右分别展示了连续型变量 x 的：中位数曲线，`median_hilow()` 所得曲线和平滑带，均值曲线，以及 `mean_cl_boot()` 所得曲线和平滑带。次行从左至右分别展示了离散型变量 x 的：`mean()` 所得均值点，`mean_cl_normal()` 所得均值点和误差棒，`median_hilow()` 所得中位数点和值域，以及 `median_hilow()` 所得中位数点和值域条。请注意 `ggplot2` 展示了整个数据的取值范围，而不仅仅是各种描述性统计量所涉及的范围。

使用 `stat_summary()` 时，你既可以为每一个参数单独地指定摘要计算函数，也可以用一个统一的函数对它们进行组合。这两种可选的方式描述如下。

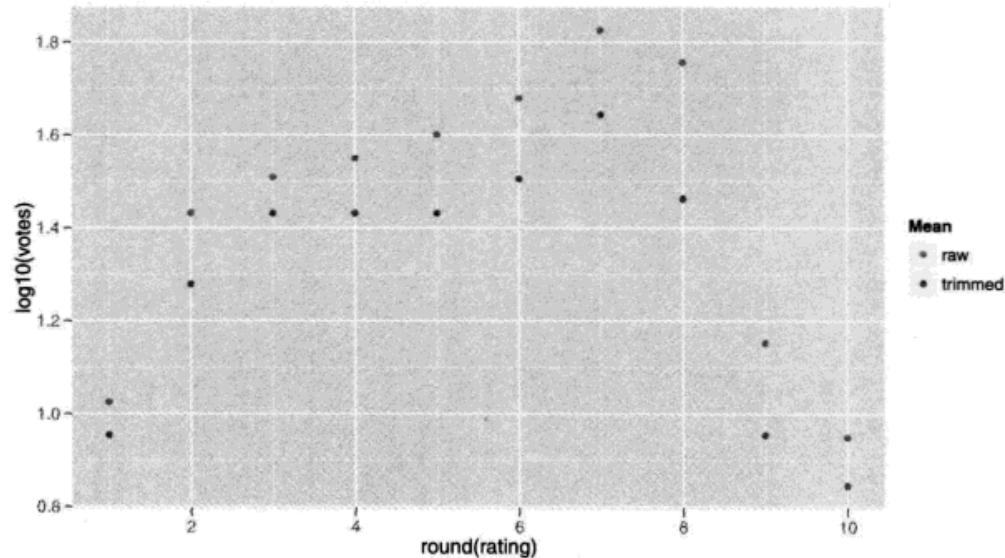
5.9.1 单独的摘要计算函数

参数 `fun.y`, `fun.ymin` 和 `fun.ymax` 能够接受简单的数值型摘要计算函数，即该函数能够传入一个数值向量并返回一个数值型结果，如：`mean()`, `median()`, `min()`, `max()`。

```

1 > midm <- function(x) mean(x, trim = 0.5)
2 > m2 +
3 +   stat_summary(aes(colour = "trimmed"), fun.y = midm,
4 +     geom = "point") +
5 +   stat_summary(aes(colour = "raw"), fun.y = mean,
6 +     geom = "point") +
7 +   scale_colour_hue("Mean")

```



5.9.2 统一的摘要计算函数

`fun.data` 可以支持更复杂的摘要计算函数，例如在表 5.3 中罗列的来自 `Hmisc` 包 (Harrell, 2008) 中的函数。当然我们也可以使用自己编写的摘要计算函数：此函数应返回一个各元素有名称的向量作为输出，如下例所示。

```

1 > iqr <- function(x, ...) {
2 +   qs <- quantile(as.numeric(x), c(0.25, 0.75), na.rm = T)
3 +   names(qs) <- c("ymin", "ymax")
4 +   qs
5 + }
6 > m + stat_summary(fun.data = "iqr", geom = "ribbon")

```

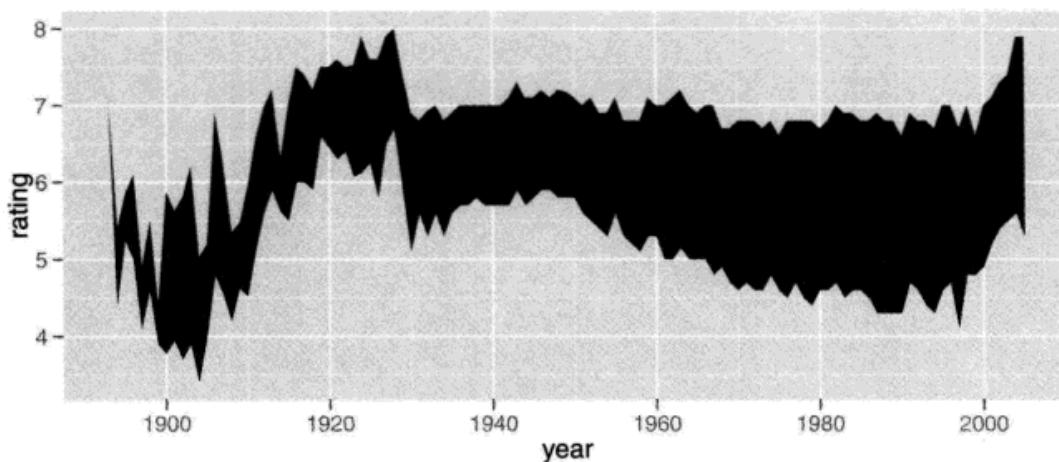


表 5.3 来自 Hmisc 包中的摘要计算函数，这些函数拥有专门的封装，以使它们能够与 `stat_summary()` 更轻松地共同使用。

函数名	Hmisc 包中原名	中间值类型	所计算区间
<code>mean_cl_normal()</code>	<code>smean.cl.boot()</code>	均值	正态渐近所得标准误
<code>mean_cl_boot()</code>	<code>smean.cl.boot()</code>	均值	Bootstrap 所得标准误
<code>mean_sdl()</code>	<code>smean.sdl()</code>	均值	标准差的倍数
<code>median_hilow()</code>	<code>smedian.hilow()</code>	中位数	尾部面积相同的外分位点对

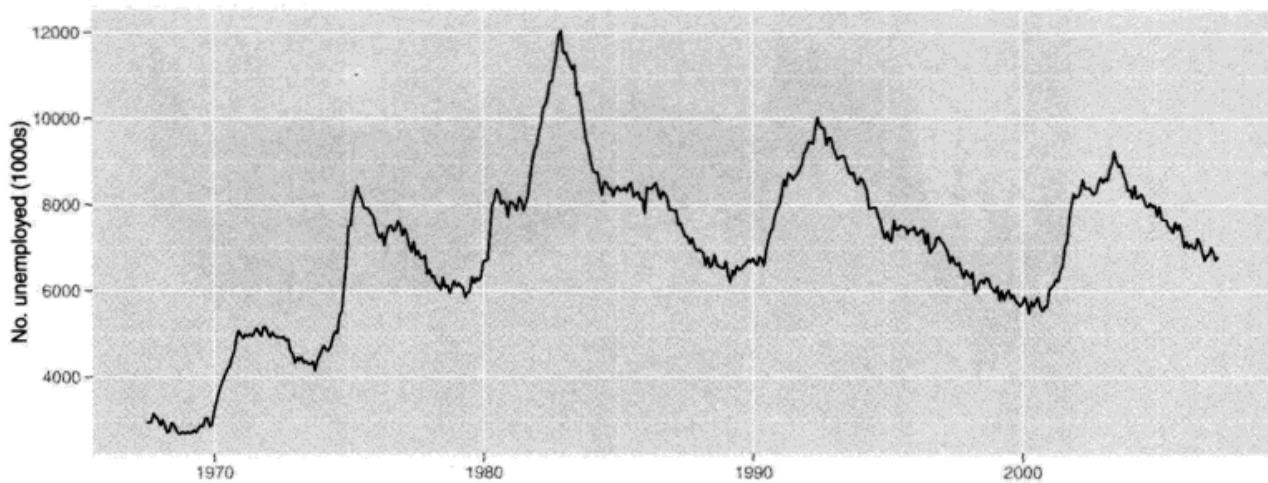
5.10 添加图形注解

在使用额外的标签注解图形时要记住的重要一点是：这些注解仅仅是额外的数据而已。添加图形注解有两种基本的方式：逐个添加或批量添加。

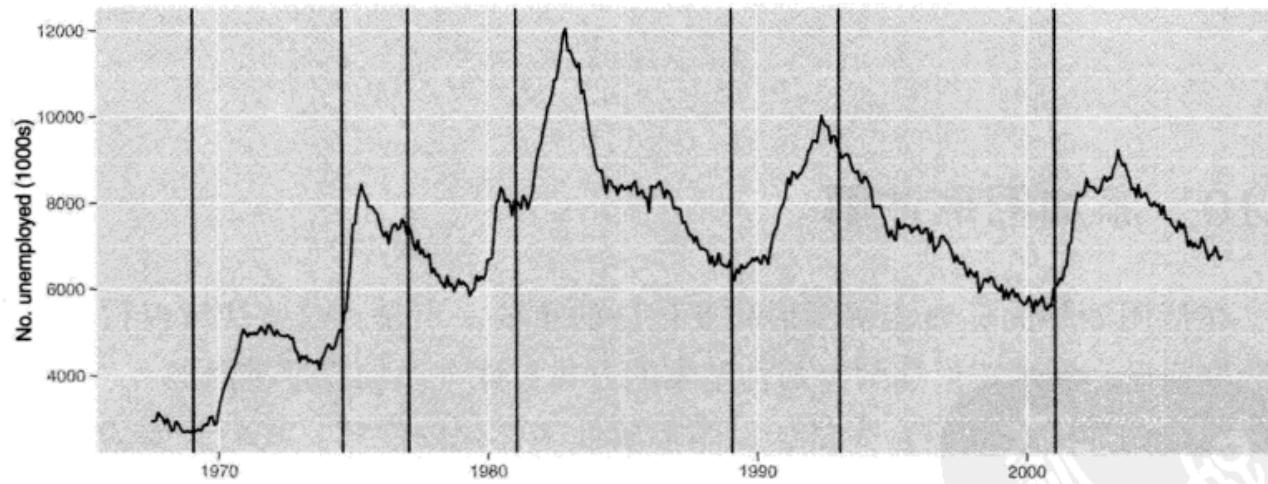
逐个添加的方式适合少量的、图形属性多样化的注解。我们只要为所有想要的图形属性设置好对应的值就可以了。如果我们需要添加多个具有类似属性的注解，将它们放到数据框中并一次添加完成也许更有效。下面的例子中，我们分别用以上两种方式^⑦，向经济数据中加入了有关美国总统的信息。

```
1 > (unemp <- qplot(date, unemployed, data = economics, geom = "line",
2 +   xlab = "", ylab = "No. unemployed (1000s)"))
```

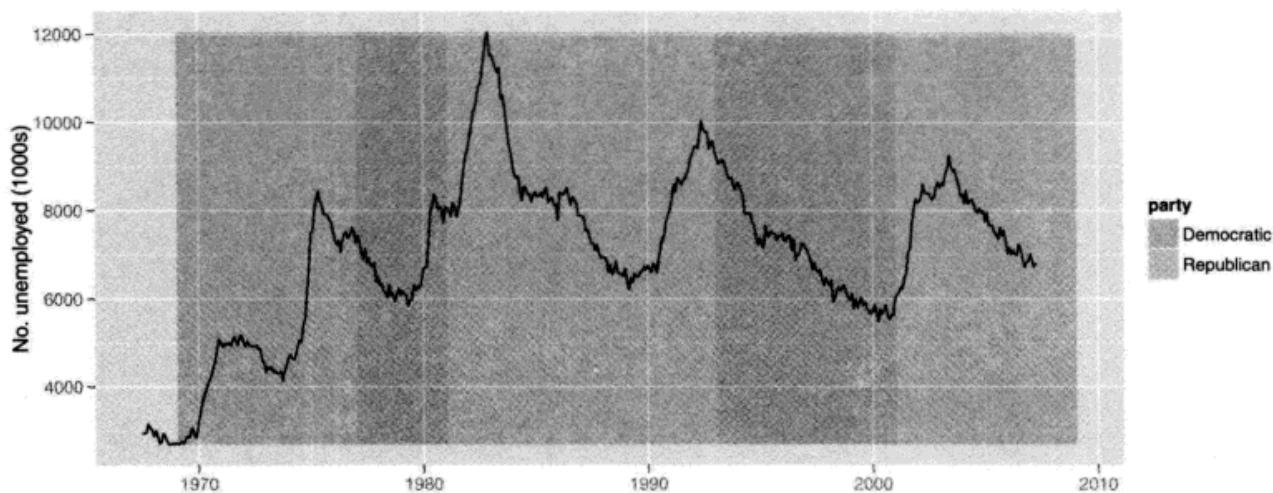
^⑦译者注：本节中第六幅图直接使用了与参数 `colour` 独立的参数 `alpha` 来指定透明度。



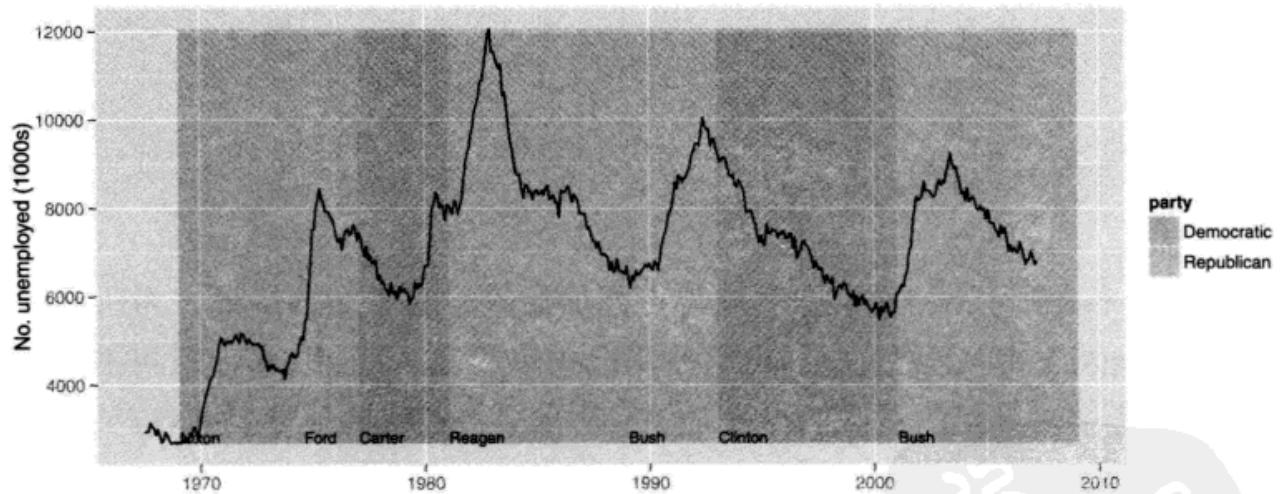
```
1 > presidential <- presidential[-(1:3), ]  
2 >  
3 > yrng <- range(economics$unemploy)  
4 > xrng <- range(economics$date)  
5 > unemp + geom_vline(aes(xintercept = as.numeric(start)), data = presidential)
```



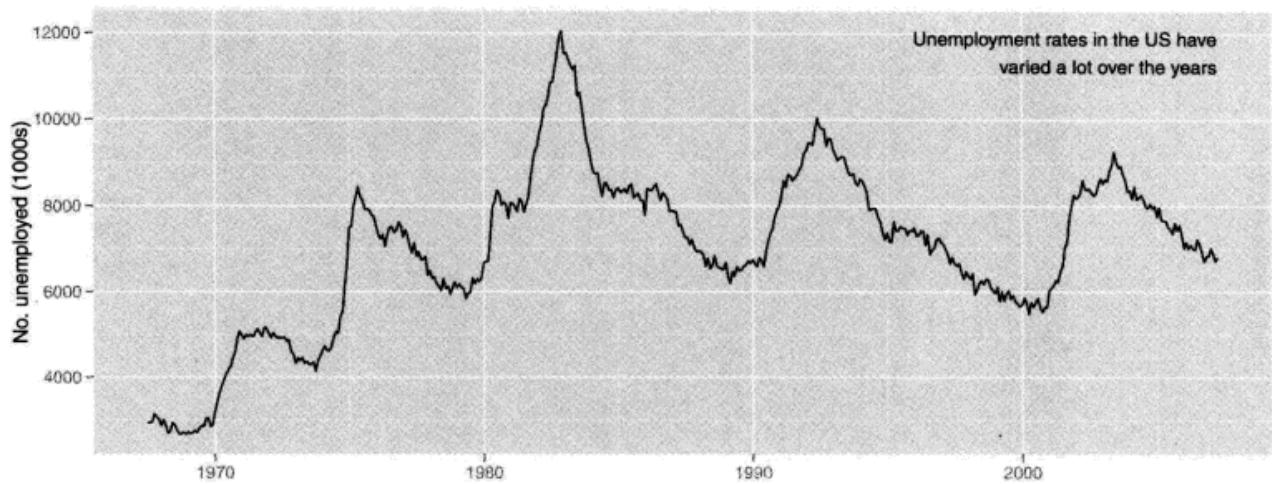
```
1 > library(scales)  
2 > unemp + geom_rect(aes(NULL, NULL, xmin = start, xmax = end,  
3 +   fill = party), ymin = yrng[1], ymax = yrng[2],  
4 +   data = presidential, alpha = 0.2) + scale_fill_manual(values =  
5 +   c("blue", "red"))
```



```
1 > last_plot() + geom_text(aes(x = start, y = yrng[1], label = name),  
2 +   data = presidential, size = 3, hjust = 0, vjust = 0)
```



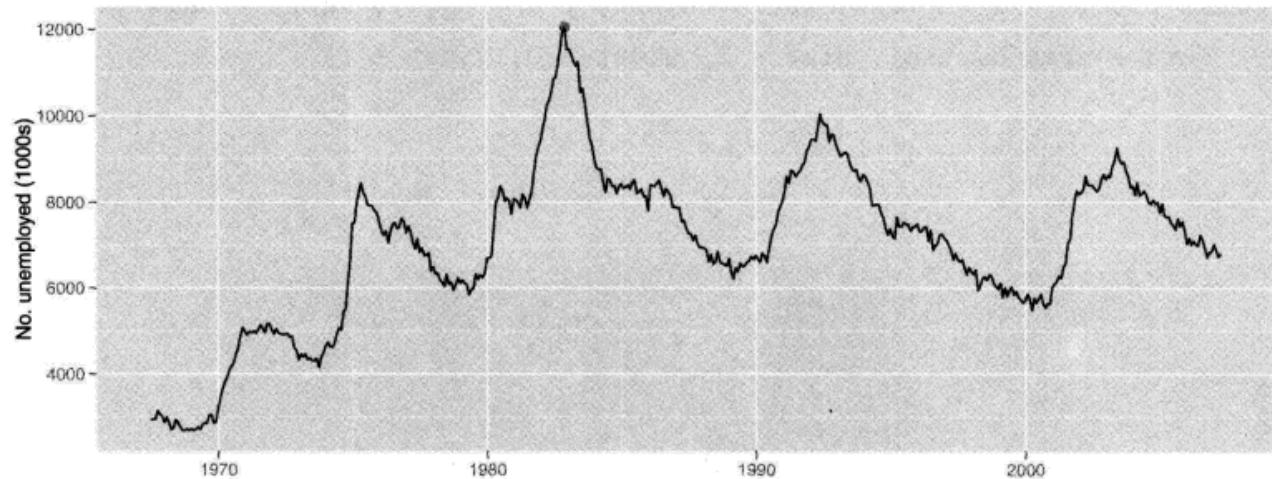
```
1 > caption <- paste(strwrap("Unemployment rates in the US have  
2 + varied a lot over the years", 40), collapse="\n")  
3 > unemp + geom_text(aes(x, y, label = caption),  
4 +   data = data.frame(x = xrng[2], y = yrng[2])),  
5 +   hjust = 1, vjust = 1, size = 4)
```



```

1 > highest <- subset(economics, unemploy == max(unemploy))
2 > unemp + geom_point(data = highest,
3 +   size = 3, colour = "red", alpha = 0.5)

```



- `geom_text` 可添加文字叙述或为点添加标签。对于多数图形，为所有观测都添加标签是无益的。然而，(使用取子集的方式) 抽取部分观测添加标签可能会非常有用——我们往往希望标注出离群点或其他重要的点。
- `geom_vline`, `geom_hline`: 向图形添加垂直线或水平线。
- `geom_abline`: 向图形添加任意斜率和截距的直线。
- `geom_rect` 可强调图形中感兴趣的矩形区域。`geom_rect` 拥有 `xmin`, `xmax`, `ymin` 和 `ymax` 几种图形属性。
- `geom_line`, `geom_path` 和 `geom_segment` 都可以添加直线。所有这些几何对象都有一个 `arrow` 参数，可以用来在线上放置一个箭头。我们也可以使用 `arrow()` 函数绘制箭头，它拥有 `angle`, `length`, `ends` 以及 `type` 几个参数。

5.11 含权数据

在处理整合后的数据 (aggregated data) 时，数据集的每一行可能代表了多种观测值，此时我们需要以某种方式把权重变量考虑进去。这里以 2000 年美国人口普查中，中西部各州的统计数据为例。此数据中主要包含的是比例型数据（例如白种人比例、贫困线以下人口比例、有大学学历的人口比例）和每个郡的基本信息（面积、人口总数、人口密度）。

有一些数据可能可以作为权重使用：

- 什么都不用，即直接观察郡的数量。
- 总人数，与原始的绝对数配合使用。
- 面积，用于研究地缘效应。

权重变量的不同将极大地影响图形内容以及观察结论。有两种可以用于表现权重的可调图形属性。首先，对于线和点这类简单的几何对象，我们可以根据点的数量调整图形属性 `size` 来改变点的大小，代码如下所示，结果见图 5.18。

```

1 qplot(percwhite, percbelowpoverty, data = midwest)
2 qplot(percwhite, percbelowpoverty, data = midwest,
3   size = poptotal / 1e6) + scale_size_area("Population\n(millions)",
4   breaks = c(0.5, 1, 2, 4))
5 qplot(percwhite, percbelowpoverty, data = midwest, size = area) +
6   scale_size_area()

```

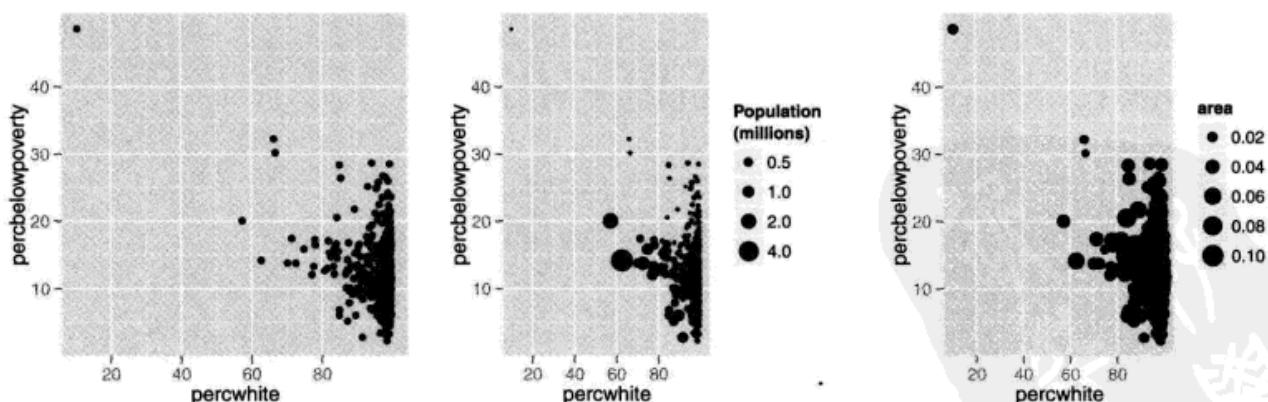


图 5.18 使用点的大小来展示权重：无权重（左图），以人口数量为权重（中图），以面积为权重（右图）。

对于更复杂的、涉及到统计变换的情况，我们通过修改 `weight` 图形属性来表现权重。这些权重将被传递给统计汇总计算函数。在权重有意义的情况下，

各种元素基本都支持权重的设定，例如：各类平滑器、分位回归、箱线图、直方图以及各类密度图。我们无法直接看到这个权重变量，而且它也没有对应的图例，但它却会改变统计汇总的结果。图 5.19 显示了作为权重的人口密度如何影响了白种人比例和贫困线以下人口比例的关系。

```

1 lm_smooth <- geom_smooth(method = lm, size = 1)
2 qplot(percwhite, percbelowpoverty, data = midwest) + lm_smooth
3 qplot(percwhite, percbelowpoverty, data = midwest,
4 weight = popdensity, size = popdensity) + lm_smooth

```

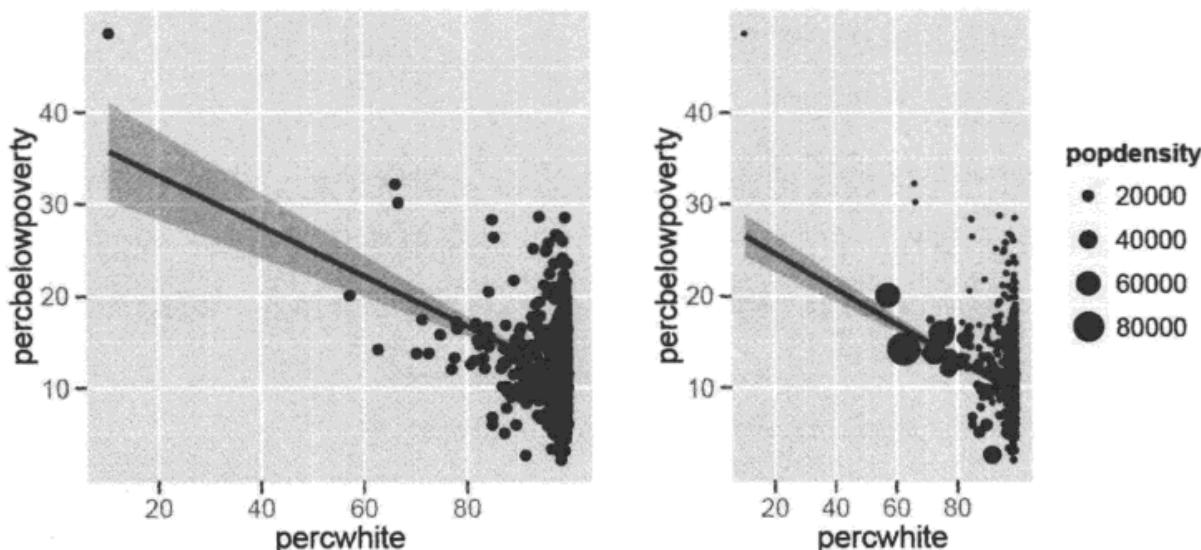


图 5.19 未考虑权重的最优拟合曲线 (左图) 和以人口数量作为权重的最优拟合曲线 (右图)。

在我们使用总人口作为权重去修改直方图或密度图的时候，我们的视角将从对郡数量分布的观察转向对人口数量分布的观察。图 5.20 以一幅贫困线以下人口比例的直方图，显示了这两种视角的不同之处。

```

1 qplot(percbelowpoverty, data = midwest, binwidth = 1)
2 qplot(percbelowpoverty, data = midwest, weight = poptotal,
3 binwidth = 1) + ylab("population")

```

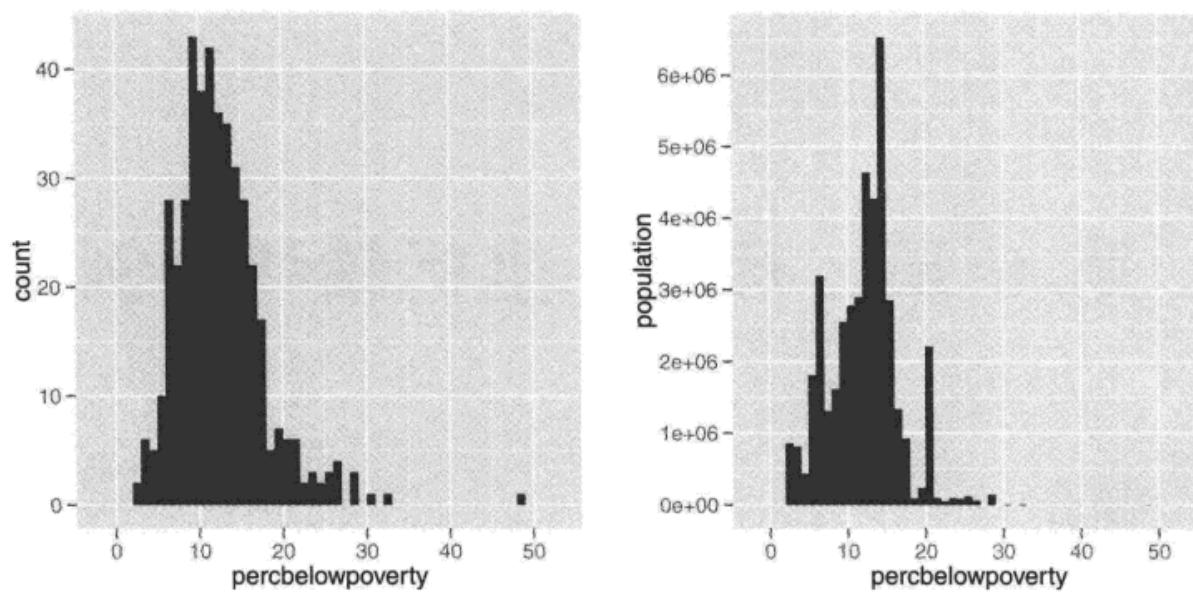


图 5.20 不含权重信息 (左侧) 以及含权重信息 (右侧) 的直方图。不含权重信息的直方图展示了郡的数量，而含权重信息的直方图展示了人口数量。权重的加入的确极大地改变了对图形的解读！

第 6 章 标度、坐标轴和图例

6.1 简介

标度 (scale) 控制着数据到图形属性的映射。标度将我们的数据转化为视觉上可以感知的东西：例如大小、颜色、位置或形状。标度也为提供了读图时所使用的工具：坐标轴和图例（总的来说可称为引导元素）。

更准确地说，每一种标度都是从数据空间的某个区域（标度的定义域）到图形属性空间的某个区域（标度的值域）的一个函数。每种标度的定义域都对应着提供给这个标度的变量的取值范围，此定义域可以是连续型或离散型、有序型或无序型。而标度的值域则包含了我们可以感知的、R 能够理解的、实实在在的图形属性：位置、颜色、形状、大小和线条类型。可能你在读到标度能够将数据同时映射为位置和颜色时会表示不解，但这是正常现象。使用同一类对象映射数据到位置和符号的这种理念，对人的直觉是一个冲击。然而，在阅读本章的更多内容后，你将体会到这种理念背后的逻辑和力量所在。

执行标度的过程分为三步：变换 (transformation)、训练 (training) 和映射 (mapping)，具体将在 6.2 节中阐述。没有标度，就没有将数据转化为图形属性的途径，所以，图形中使用的每一个图形属性都需要一个标度。当然，如果我们每次使用一个新的图形属性时都去手动添加一个标度，又难免枯燥乏味，所以当有需要时，ggplot2 将自动添加一个默认的标度。我们确实可以在不了解标度运作原理的情况下画出许多图形，但理解标度并学会如何操纵它们则将赋予我们对图形更强的控制能力。关于默认标度以及修改它们的方法，详见 6.3 节。

标度可以粗略地分为四类：位置标度、颜色标度、手动离散型标度以及同型标度。常用选项和主要使用方法在 6.4 节进行了描述。这一节着重给出对

可用选项的总体概览，而非深入地逐个展开细节。各参数的更多细节已记录于在线文档中。

标度的另一个重要角色，是生成一个允许读图者从图形属性空间到数据空间进行反向映射的引导元素 (**guide**)，并从图中读出取值。对于位置型图形属性，引导元素是坐标轴；对于所有其他图形属性来说，引导元素是图例。与其他图形系统不同的是，在这里我们基本无法直接控制坐标轴和图例的细节——不存在 `gglegend()` 或者 `ggaxis()` 这种修改图例或坐标轴的函数^①。事实上，引导元素的外观皆由标度的参数控制。6.5 节中讨论了坐标轴和图例。

6.6 节征引了部分探讨图形属性赋值问题的学术著作，为本章作结。

6.2 标度的工作原理

要阐明标度的工作原理，首先需要了解标度的定义域（即数据空间）和标度的值域（即图形属性空间），然后说明两者之间进行映射的过程。

由于输入变量可能是离散型，也可能是连续型，所以标度的定义域要么是某些值组成的集合（以因子的形式存储，字符型因子或逻辑型因子），要么就是一个实值区间（以长度为 2 的数值型向量的形式存储）。以哺乳动物睡眠数据 (`msleep`) 为例，离散型变量 `vore` 的定义域为 `{carni, herbi, omni}`，连续型变量 `bodywt` 的定义域为 `[0.005, 6654]`。我们通常会认为这些量是数据的“值域”，但在这里，我们关注的是它们作为标度输入的本质——一个函数的定义域。

标度的值域也可以是离散型或连续型的。对于离散型标度，它的值域是输入值对应的图形属性值组成的一个向量。对于连续型标度，它的值域是穿过某种更复杂空间的一条一维路径。例如，从一种颜色到另一种颜色进行线性插值得到的一组渐变颜色。标度的值域要么在其被创建时由用户指定，要么由这个标度自行指定。

将定义域映射到值域的过程包括以下阶段：

- **变换：**（仅针对连续型的定义域）对数据进行变换后再进行展示往往是有益的，例如对数据取对数或开根号。6.4.2 节对变换进行了更加深入地探讨。
在变换完成后，将会基于变换后的数据计算每一层的统计摘要。此举保证

^①译者注：0.9.0 版本以后，`guides()` 以及其他相关 `guide_XXX` 一系列的函数增强了对图例 (legend) 的直接控制。

了线性尺度上 $\log(x)$ 对 $\log(y)$ 的图形与对数尺度上 x 对 y 的图形看起来相同。

- **训练：**在这个关键的阶段，将通过学习 (learning) 得到标度的定义域。有时，学习某个标度的定义域相当简单直接：在一个仅有一个图层且仅呈现原始数据的图形中，这个学习过程包括确定某个 (变换后的) 连续型变量的最小值和最大值，或者是列出某个类别型变量的所有水平。但是，标度的定义域往往必须在多个面板 (panel) 中反映出横跨多个数据集的多个图层。举例来说，试想这样一个标度，它将被用来创建一个坐标轴；而第一层的原始数据的最大最小值与第二层中的统计摘要有可能是不同的，但是它们最后必须都被绘制在图形上。

如 6.3 节所述，定义域亦可通过手动设定控制标度值域的参数 `limits` 跳过训练过程而被直接指定。在此标度定义域外的任何值都将被映射为 NA。

- **映射：**我们现在已经了解了标度的定义域，并且在整个过程开始之前就已经了解了标度的值域，最后要做的一件事，就是执行这个映射数据到图形属性的函数。

简洁起见，我们在描述这个过程时省略了一些步骤。比如，我们没有讨论分面在训练过程中的角色，同时也忽略了对位置的调整。即使如此，这个概述还是很准确的，当你对各种标度在图形中的表现感到困惑时，你应该回过头来看看这个概述。

6.3 用法

每一种图形属性都拥有一个默认的标度，此标度将在我们每次使用这个图形属性时被自动添加到图形中。这些标度列于表 6.1 中。标度的种类依赖于变量的类型：标度可为连续型 (变量为数值时) 或离散型 (变量为因子、逻辑值、字符串时)。要修改默认的标度，参看 8.2.1 节中所述的函数 `set_default_scale()`。

在我们初始化整个图形和增加新图层时，默认的标度将被自动添加。这意味着，如果我们在之后修改了底层数据或图形属性映射，变量类型和标度类型之间可能出现不匹配的情况。当这种情况发生时，我们需要自行添加正确的标度。下例演示了这个问题及其解决方法^②。

^②译者注：较新版本的 `ggplot2` 可以自动完成变换。

表 6.1 按图形属性和变量类型排列的各种标度。默认标度以粗体显示。默认的标度根据变量是连续型或是离散型而有所不同。形状 (shape) 和线条类型 (line type) 没有默认的连续型标度；大小 (size) 没有默认的离散型标度。

图形属性	离散型	连续型
颜色 (colour) 和填充色 (fill)	brewer grey hue identity manual	gradient gradient2 gradientn
位置 (position)(x, y)	discrete	continuous date
形状 (shape)	shape identity manual	
线条类型 (line type)	linetype identity manual	
大小 (size)	identity manual	size

```

1 plot <- qplot(cty, hwy, data = mpg)
2 plot
3
4 ## 这样做行不通是因为变量类型和默认标度不匹配
5 plot + aes(x = drv)
6
7 ## 更正默认标度后解决了问题.
8 plot + aes(x = drv) + scale_x_discrete()

```

如果要添加一个不同的标度或修改默认标度的某些特征，我们必须构造一个新的标度，然后使用 `+` 将其添加到图形上。所有的标度构建器 (scale constructor) 都拥有一套通用的命名方案。它们以 `scale_` 开头，接下来是图形属性的名称（例如，`colour_`、`shape_` 或 `x_`），最后以标度的名称结尾（例如，`gradient`、`hue` 或 `manual`）。举例来说，离散型数据的颜色图

形属性的默认标度名为 `scale_colour_hue()`，填充色的 Brewer 配色标度名为 `scale_fill_brewer()`。

以下代码^③阐明了这个过程。我们以一个使用默认颜色标度的图形开始，然后修改其图例的外观，接着使用了一种不同的颜色标度。结果如图 6.1 所示。

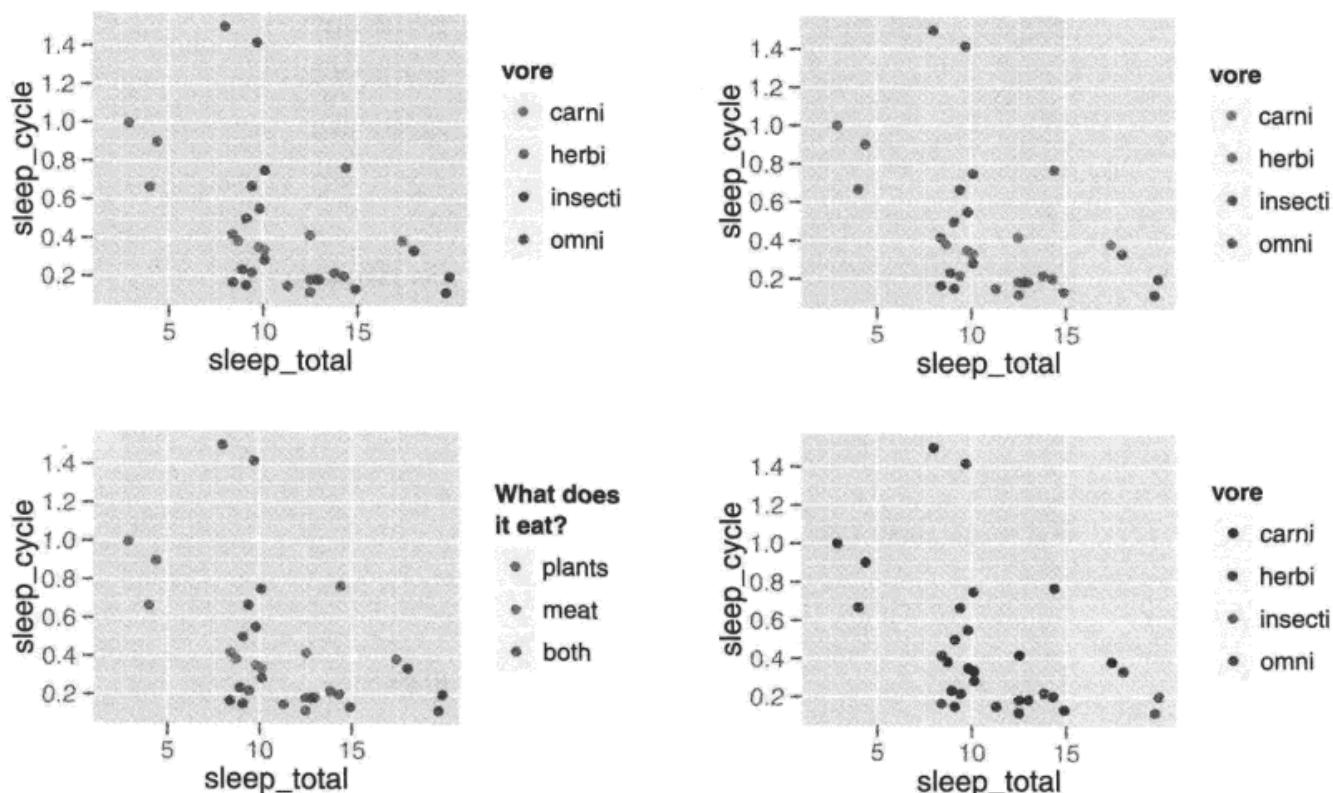


图 6.1 (参见彩图 19) 调整标度默认参数的示例。(左上图) 使用默认标度的图形。(右上图) 手动添加默认标度，并未改变图形外观。(左下图) 调整标度的参数以实现对图例的微调。(右下图) 使用一种不同的颜色标度：ColorBrewer 配色方案中的 Set1。

```

1 p <- qplot(sleep_total, sleep_cycle, data = msleep, colour = vore)
2 p
3 ## 显式添加默认标度
4 p + scale_colour_hue()
5
6 ## 修改默认标度的参数，这里改变了图例的外观
7 p + scale_colour_hue("What does\nit eat?",
8   breaks = c("herbi", "carni", "omni", NA),
9   labels = c("plants", "meat", "both", "don't know"))
10

```

^③译者注：最后一幅图原始代码中函数 `scale_fill_brewer()` 的参数 `pal` 在新版本中名为 `palette`。

```
11 ## 使用一种不同的标度  
12 p + scale_colour_brewer(palette = "Set1")
```

6.4 标度详解

标度可大致分为四组：

- 位置标度，用于将连续型、离散型和日期 – 时间型变量映射到绘图区域，以及构造对应的坐标轴。
- 颜色标度，用于将连续型和离散型变量映射到颜色。
- 手动标度，用于将离散型变量映射到我们选择的符号大小、线条类型、形状或颜色，以及创建对应的图例。
- 同一型标度，用于直接将变量值绘制为图形属性，而不去映射它们。举例来说，假设我们想要将变量映射为符号的颜色，而此变量本身就是一个由颜色值组成的向量，那么我们就无须再将其映射为其他的颜色，直接渲染这些值本身即可。

本节阐述了每组标度的更多细节。各独立标度更精确的细节可在 R 中的文档内找到（例如，`?scale_brewer`），或阅读在线文档 <http://had.co.nz/ggplot2>。

6.4.1 通用参数

以下参数对所有标度通用。

- `name`：设置坐标轴或图例上出现的标签。可以指定字符串（使用\n换行）或数学表达式（语法如 `?plotmath` 所述）。由于经常需要微调这些标签，所以使用三个辅助函数 `xlab()`、`ylab()` 和 `labs()` 可以让我们减少部分键入。以下代码演示了它们的用法，结果见图 6.2。

```
1 p <- qplot(cty, hwy, data = mpg, colour = displ)  
2 p  
3 p + scale_x_continuous("City mpg")  
4 p + xlab("City mpg")  
5 p + ylab("Highway mpg")
```

```
6 p + labs(x = "City mpg", y = "Highway", colour = "Displacement")
7 p + xlab(expression(frac(miles, gallon)))
```

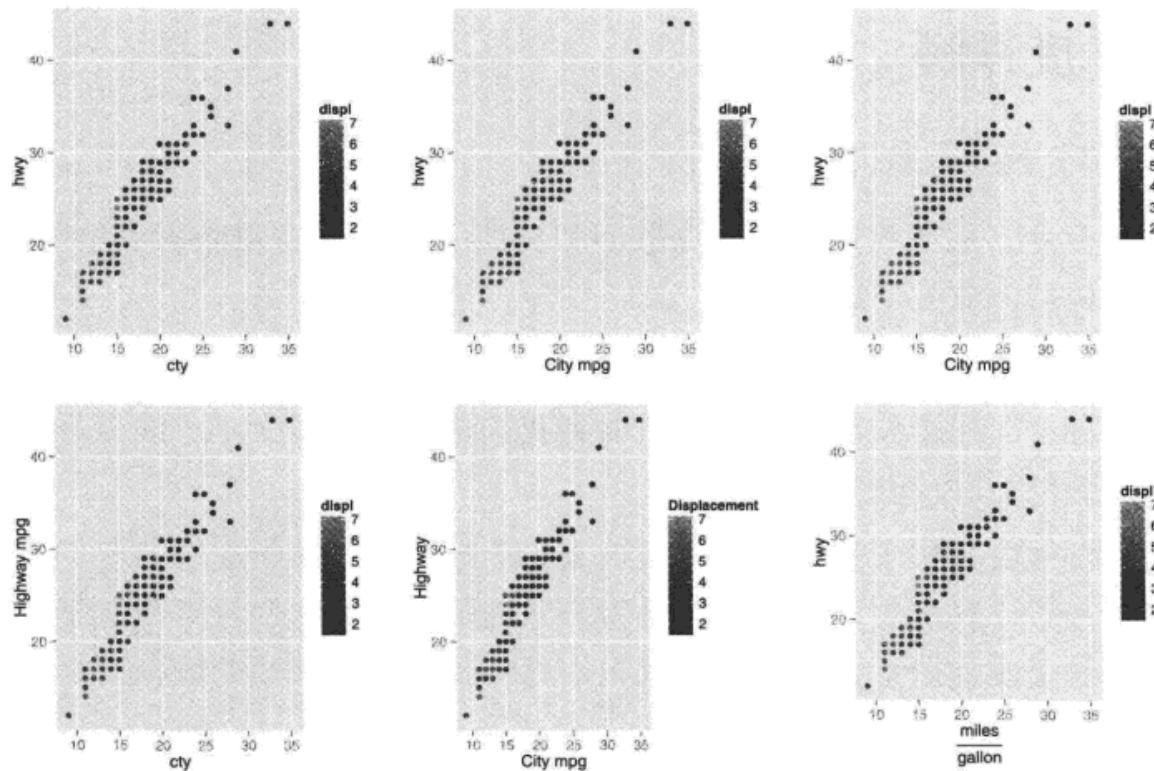


图 6.2 图例名称可以接受不同形式的参数。

- **limits:** 固定标度的定义域。连续型标度接受一个长度为 2 的数值型向量；离散型标度接受一个字符型向量。一旦设定了 **limits**，数据将不再进行任何训练。读者可直接跳到 6.4.2 节以了解更多。限制定义域可以帮助我们移除不想在图形上展示的数据（即设置比数据的完整范围更小的 **limits**），同时也可以保证要进行比较的多个图形中的绘制范围一致（即设置比默认范围大一些或小一些的 **limits**）。

任何不在此标度定义域内的值均将被丢弃：如果想要囊括图中的所有观测，其每个图形属性都必须位于每个标度的定义域中。丢弃过程发生在统计量的计算之前。

- **breaks** 和 **labels**: **breaks** 控制着显示在坐标轴或图例上的值，即，坐标轴上应该显示哪些刻度线的值，或一个连续型标度在一个图例中将被如何分段。**labels** 指定了应在断点处显示的标签。若设置了 **labels**，则必须同时指定 **breaks**，只有这样，这两个参数才能被正确匹配。

要区分 **breaks** 和 **limits**，请牢记，**breaks** 影响显示在坐标轴和图例上

的元素，而 `limits` 影响显示在图形上的元素，见图 6.3。图中第一列使用了 `breaks` 和 `limits` 的默认设置，即 `limits = c(4, 8)` 和 `breaks = 4:8`。中间一列重新设置了 `breaks`: 绘图区域不变，但刻度和标签的位置都被移动了。最右列则重新定义了 `limits`，相当多的数据落在了绘图区域之外^④。

```
1 p <- qplot(cyl, wt, data = mtcars)
2 p
3 p + scale_x_continuous(breaks = c(5.5, 6.5))
4 p + scale_x_continuous(limits = c(5.5, 6.5))
5 p <- qplot(wt, cyl, data = mtcars, colour = cyl)
6 p
7 p + scale_colour_gradient(breaks = c(5.5, 6.5))
8 p + scale_colour_gradient(limits = c(5.5, 6.5))
```

- **formatter:** 如果未指定任何标签，则将在每个断点处自动调用格式刷 (`formatter`) 来格式化生成标签。对于连续型标度，可用的标签刷为：`comma`, `percent`, `dollar` 和 `scientific`; 对于离散型标度，则为 `abbreviate`。

6.4.2 位置标度

每幅图形一定拥有两个位置标度，一个指定水平位置 (x 标度)，另外一个是竖直位置 (y 标度)。`ggplot2` 提供了连续型、离散型 (针对因子型、字符型和逻辑型向量) 以及日期型标度。其中每种标度对数据作变换的方式稍有不同，并各自生成一个类型稍有不同的坐标轴。以下几节针对每种类型作了详述。

对所有的位置坐标来说，修改坐标轴的范围是一项常见任务。由于这项任务需要被经常执行，所以 `ggplot2` 提供了一对辅助函数 `xlim()` 和 `ylim()` 来减轻我们的工作量。这些辅助函数会检查它们的输入，然后创建合适的对应标度，如下所示：

- `xlim(10, 20)`: 一个从 10 到 20 的连续型标度；
- `ylim(20, 10)`: 一个从 20 到 10 的反转后连续型标度；

^④译者注：0.9.0 版本以后，所有标度 (`scales`) 可以接受 `na.value` 参数，用来将数据中的 NA 值映射到合适的图形属性，默认的颜色 (`color`) 和填充 (`fill`) 的值为灰色，`na.value` 等于 NA 时为透明。本例涉及到 `ggplot2` 中处理限定范围 (`range`) 外数据的方法，方法在未来更新的版本可能会有变化，请随时关注发布日志。0.9.1 将颜色限制范围外的数据处理为缺失值 (`missing value`)，所以可通过 `na.value` 控制其图形属性。

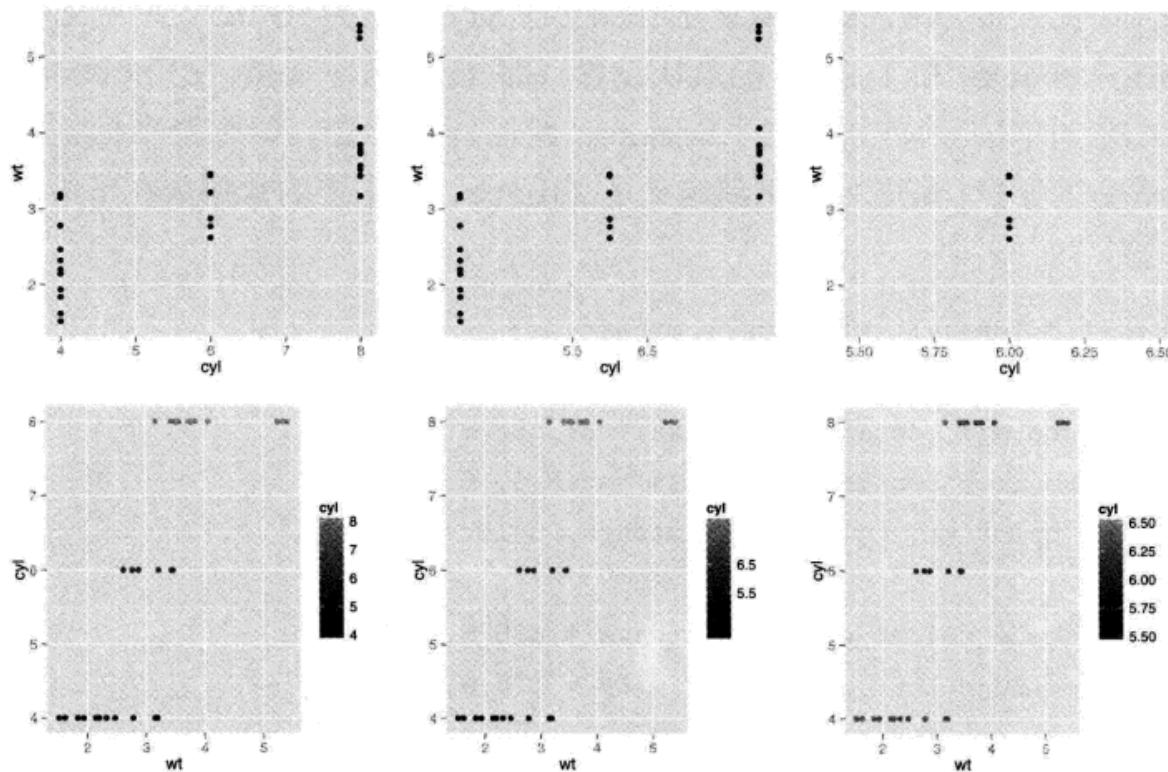


图 6.3 `breaks` 和 `limits` 的区别。对 x 轴的影响 (首行) 以及彩色图例 (次行)。(左栏) 默认参数 `limits = c(4, 8)`, `breaks = 4:8` 的图形, (中栏)`breaks = c(5.5,6.5)`, (右栏) `limits = c(5.5,6.5)`。

- `xlim("a", "b", "c")`: 一个离散型标度;
- `xlim(as.Date(c("2008-05-01", "2008-08-01")))`: 一个从 2008 年 5 月 1 日到 8 月 1 日的日期型标度。

以上参数与 R 基础图形和 lattice 图形中的参数 `xlim` 和 `ylim` 的工作方式不同。在 ggplot2 中, 为了保持与其他标度的一致性, 任何在 `limits` 以外的数据都不会被绘制, 也不会被包括在统计变换的过程中。这就意味着, 通过设置 `limits` 所得结果与在视觉上放大一块绘图区域所得的结果是不同的。要实现后者, 我们需要使用 `coord_cartesian()` 函数的参数 `xlim` 和 `ylim`, 在 7.3.3 节有论述。后一种方式单纯地对图形进行了视觉上的放大, 而不影响底层的数据。

默认情况下, 位置标度的 `limits` 会稍微超出数据的范围。这样就保证了数据与坐标轴不会发生重叠。我们可以使用参数 `expand` 来控制溢出量, 此参数是长度为 2 的数值型向量。其中第一个元素给出的是乘式的溢出量,

第二个参数给出的是加式的溢出量。如果你不想留任何多余的空间，就使用 `expand = c(0, 0)`。

连续型

最常用的连续型位置标度是 `scale_x_continuous` 和 `scale_y_continuous`，它们均将数据映射到 x 轴和 y 轴。而最有趣的变式是通过变换来生成的。每个连续型标度均可接受一个 `trans` 参数，允许指定若干种线性或非线性的变换。而每一种变换都是由所谓的“变换器”(transformer) 来实现的，变换器描述了变换本身和对应的逆变换，以及如何去绘制标签。表 6.2 列出了部分较为常用的变换器。

表 6.2 内建的变换器列表。

名称	变换函数 $f(x)$	逆变换函数 $f^{-1}(y)$
asn	$\tanh^{-1}(x)$	$\tanh(y)$
exp	e^x	$\log(y)$
identity	x	y
log	$\log(x)$	e^y
log10	$\log_{10}(x)$	10^y
log2	$\log_2(x)$	2^y
logit	$\log(\frac{x}{1-x})$	$\frac{1}{1+e(y)}$
pow10	10^x	$\log_{10}(y)$
probit	$\Phi(x)$	$\Phi^{-1}(y)$
recip	x^{-1}	y^{-1}
reverse	$-x$	$-y$
sqrt	$x^{1/2}$	y^2

变换通常被用来修改位置标度，所以对于 x, y 和 z 标度都是有简便写法的：`scale_x_log10()` 与 `scale_x_continuous(trans = "log10")` 是等价的。参数 `trans` 对任意的连续型标度均有效，包括下文中描述的颜色梯度，但简便写法仅针对位置标度存在。

当然，我们也可以自行实施变换。举例来说，我们可以直接绘制 `log10(x)`，而不去使用 `scale_x_log()`。这两种做法将在绘图区域生成完全相同的结果，但坐标轴和刻度标签却是不同的。如果我们使用变换后的标度，坐标轴将依原

始的数据空间进行标注。在这两种情况下，变换均在计算统计摘要之前发生。使用以下代码绘制的图 6.4 演示了两者的不同。

```
1 qplot(log10(carat), log10(price), data = diamonds)
2 qplot(carat, price, data = diamonds) +
3   scale_x_log10() + scale_y_log10()
```

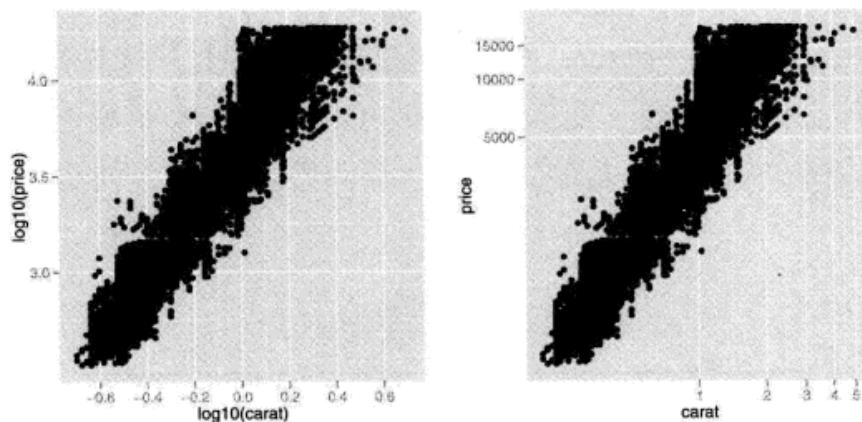


图 6.4 钻石价格和重量的散点图。演示了对标度进行对数变换（左图）和对数据进行对数变换（右图）的异同。图形主体是完全相同的，但坐标轴上的标签是不同的。

变换器同样可以用在 `coord_trans()` 中，此时变换将在统计量计算完成后进行，因此将影响图形主体的外观。函数 `coord_trans()` 在 7.3.3 节有详述。

日期和时间

日期和时间值基本上属于连续型，但在标注坐标轴时有着特殊的处理方式。目前我们仅支持属于 `date` 类的日期值和属于 `POSIXct` 类的时间值。如果你的日期和时间值是其他格式的，则需使用 `as.Date()` 或 `as.POSIXct()` 对其进行转换。

对于日期坐标轴，有三个参数可用于控制其外观和刻度的位置：`major`、`minor` 以及 `format`。总体而言，此标度本身已经能够很好地选择默认值，不过如果你想对其进行微调，细节如下：

- 参数 `major` 和 `minor` 用以按照时间的单位，即年 (year)、月 (month)、周 (week)、日 (day)、时 (hour)、分 (minute)、秒 (second) 来指定主要和次要断点的位置，并且允许以这些单位的倍数出现。例如，`major = "2 weeks"` 将在每隔两周的位置放置一个主刻度。如果未被指定，日期标度可以自动选出合适的默认值。

- 参数 `format` 指定了刻度标签的格式。表 6.3 列出了用以展示日期各个组成部分的特殊字符。例如，如果你想以 14/10/1979 的形式显示日期，就可以使用字符串 "%d/%m/%y"。

表 6.3 从 `strptime` 的文档中提取的常用数据格式编码。依时间从短到长排列。

编码	含义
%S	秒 (00–59)
%M	分钟 (00–59)
%I	小时, 12 小时制 (1–12)
%H	小时, 24 小时制 (00–23)
%a	缩写的周几 (Mon–Sun)
%A	全称的周几 (Monday–Sunday)
%e	某月中的某天 (1–31)
%d	某月中的某天 (01–31)
%m	以数值表示的月份 (01–12)
%b	缩写的月份 (Jan–Dec)
%B	全称的月份 (January–December)
%y	不含世纪的年份 (00–99)
%Y	含世纪的年份 (0000–9999)

我们通过绘制图 6.5 来演示部分参数的用法，代码如下^⑤。

```

1 library(scales)
2 plot <- qplot(date, psavert, data = economics, geom = "line") +
3   ylab("Personal savings rate") +
4   geom_hline(xintercept = 0, colour = "grey50")
5 plot
6 plot + scale_x_date(breaks = date_breaks("10 years"))
7 plot + scale_x_date(
8   limits = as.Date(c("2004-01-01", "2005-01-01")),
9   labels = date_format("%Y-%m-%d")
10 )

```

^⑤译者注：第二幅图原始代码中使用的参数 `major` 已经失效，这里我们载入了 `scales` 包并使用其中的函数 `date_breaks()` 为新参数 `breaks` 生成了坐标轴的断点。同时，第三幅图原始代码中的参数 `format` 也已经失效，我们需要使用 `labels = date_format(...)` 为坐标轴生成标签。

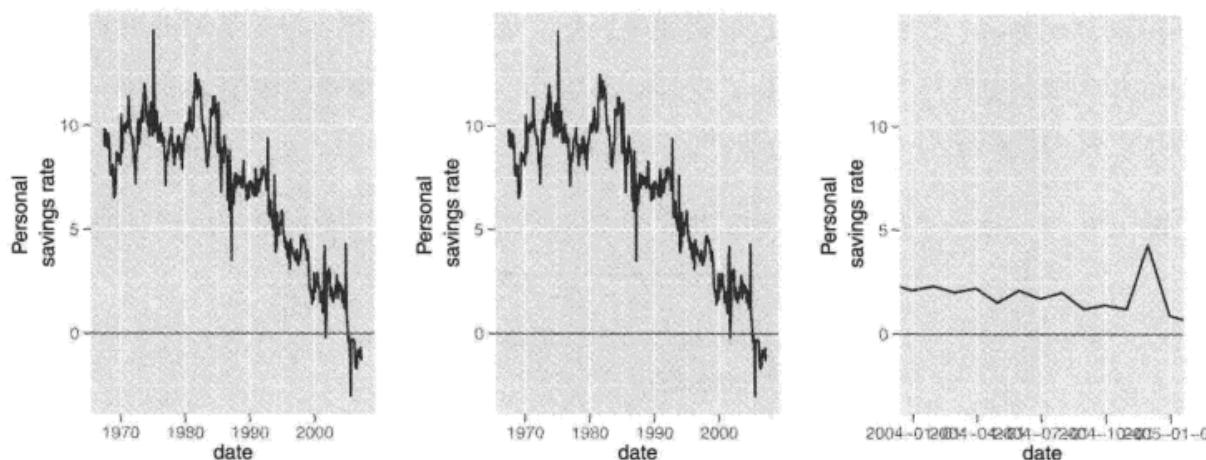


图 6.5 个人储蓄率的时间序列图形。(左图) 默认外观, (中图) 每隔 10 年为一断点, 以及(右图) 使用年月日的格式仅显示在 2004 年内的图形。测量值是在每月月末记录的。

离散型

离散型位置标度将输入中的各水平映射为整数。结果的顺序可用参数 `breaks` 进行控制, 不想要的水平可以使用 `limits`(或使用 `xlim()` 或 `ylim()`) 进行丢弃。由于我们经常也会在图形的非整点位置放置标签和标注, 所以离散型位置标度也可以接受连续型的值。如果你尚未调整 `breaks` 或 `limits`, 某个因子水平的所在位置的数值表示可以使用 `as.numeric()` 进行计算: 以从 1 开始的整数表示。

6.4.3 颜色标度

除了位置标度以外, 最常用的图形属性可能就是颜色了。有相当多的方法用来将值映射为颜色: 对连续型值有三种基于渐变的方法, 对离散型值有两种方法。但在我们研究每种方法的细节之前, 先了解一点色彩学理论是很有帮助的。正因为人眼和大脑的底层生理学原理比较复杂, 所以色彩学也较为复杂, 下面的简介将着重强调那些相对重要的问题。一份更详细的优秀说明可在 <http://tinyurl.com/clrdtls> 找到。

在物理层面, 颜色是由不同波长的光混合而产生的。要完全了解一种颜色, 我们需要了解全部的波长混合情况, 但幸运的是, 我们人类的眼球只有三种不同的颜色感受器, 于是我们就可以仅用三个数字来表示任意颜色。你可能对于使用 `rgb` 编码的色彩空间已经很熟悉了, 这种色彩空间使用红、绿、蓝三种光的光强来表示一种颜色。这个空间的一个问题在于, 它在视觉感知上并不均匀:

两种间隔一个单位的颜色可能看上去非常相似，但又可能非常不同，这取决于它们在这个色彩空间中的位置。这使得创建一个从连续变量到一个颜色集的映射变得十分困难。在提出感知上更均匀的色彩空间方面已经有了很多尝试。我们在这里将使用一种名为 hcl 色彩空间的现代方案，它由三部分构成，分别是色相 (hue)、彩度 (chroma) 和明度 (luminance)：

- 色相 (hue) 是一个 0 和 360 之间的 (角度) 值，它将一种色彩赋以“颜色”属性：如蓝、红、橙等等。
- 明度 (luminance) 指颜色的明暗程度。明度的高低，要看其接近白色或黑色的程度而定。明度为 0 为黑，明度为 1 为白。
- 彩度 (chroma) 指色彩的纯度。彩度为 0 是灰色，彩度的最大值随明度的变化而不同。

这三部分组合生成的空间形状并不简单。图 6.6 尝试展示了此色彩空间的三维形状。每个分面中的明度 (亮度) 是一个常数，色相被映射为角度，彩度被映射为半径。我们可以看到，每个分面的中心均为灰色，距离边缘越近，颜色也越浓烈。

另一个问题是，很多人 (约为人群中的 10%) 不具有功能健全的颜色感受器，因而只能分辨出相对更少的颜色。简言之，最好避免使用红——绿对比，并且要使用可以模拟色盲情形的系统来检查你的图形。Visicheck 就是一种在线解决方案。另外一种选择是使用 `dichromat` 包 (Lumley, 2007)，它提供了模拟色盲情形的工具，并且自带了若干套已知对色盲人士亦可正常工作的配色方案。同样地，你还可以像帮助色盲人士一样帮助只拥有黑白打印机的人们：提供到其他图形属性 (如大小、线条类型或形状) 的冗余映射。

以下几节讨论的所有标度对边界色 (`colour`) 图形属性和填充色 (`fill`) 图形属性均有效。

连续型

根据颜色梯度中的色彩数量划分，共有三类连续型颜色梯度 (即渐变色)：

- `scale_colour_gradient()` 和 `scale_fill_gradient()`: 双色梯度。顺序为从低到高。参数 `low` 和 `high` 用以控制此梯度两端的颜色。
- `scale_colour_gradient2()` 和 `scale_fill_gradient2()`: 三色梯度。顺

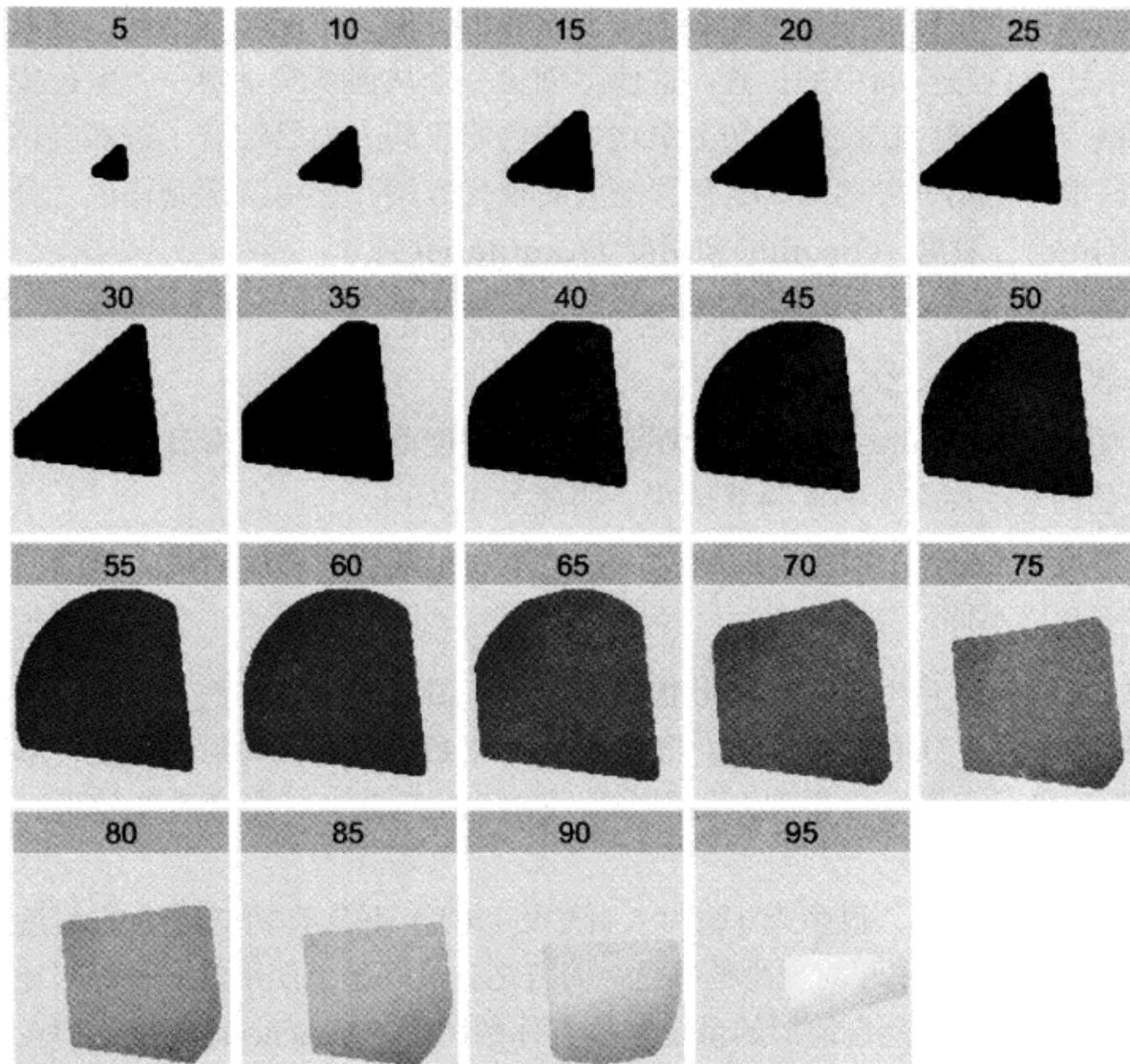


图 6.6 (参见彩图 20)hcl 色彩空间的形状。色相被映射为角度，彩度被映射为半径，每个分面各自表示一种明度。hcl 色彩空间的形状比较奇异，但我们也能够看出每个分面的中心为灰色，随着向边缘移动，颜色变得愈加浓烈。明度为 0 和 100 的切片已被省略，因为它们分别是一个黑点和一个白点。

序为低——中——高。参数 `low` 和 `high` 作用同上，这两种标度还在中点处拥有一个中间色。中点的默认值为 0，但也可使用参数 `midpoint` 将其设置为任意值。这个参数对生成发散型 (diverging) 配色方案特别有用。

- `scale_colour_gradientn()` 和 `scale_fill_gradientn()`: 自定义的 n- 色梯度。此标度需要赋给参数 `colours` 一个颜色向量。不加其他参数的话，这些颜色将依照数据的范围均匀地分布。如果你需要让这些值不均匀地分布，则可以使用参数 `values`。如果 `rescale` 的取值为 `TRUE`(默认

值), 则 `values` 应在 0 和 1 之间取值, 如果 `rescale` 的取值为 `FALSE`, 则 `values` 应在数据范围内取值。

颜色梯度常被用来展示一个二维表面的高度。下面我们将以 `faithful` 数据集 (Azzalini and Bowman, 1990) 的一个二维核密度估计表面为例进行说明, 此数据集记录了黄石公园的老忠实泉两次喷发的间隔时间以及每次喷发的持续时长。图 6.7 展示了应用于此数据的三种颜色梯度, 绘图代码如下。注意 `limits` 的用法: 此参数对所有标度都是通用的。

```

1 f2d <- with(faithful, MASS::kde2d(eruptions, waiting,
2   h = c(1, 10), n = 50))
3 df <- with(f2d, cbind(expand.grid(x, y), as.vector(z)))
4 names(df) <- c("eruptions", "waiting", "density")
5 erupt <- ggplot(df, aes(waiting, eruptions, fill = density)) +
6   geom_tile() +
7   scale_x_continuous(expand = c(0, 0)) +
8   scale_y_continuous(expand = c(0, 0))
9 erupt + scale_fill_gradient(limits = c(0, 0.04))
10 erupt + scale_fill_gradient(limits = c(0, 0.04),
11   low = "white", high = "black")
12 erupt + scale_fill_gradient2(limits = c(-0.04, 0.04),
13   midpoint = mean(df$density))

```

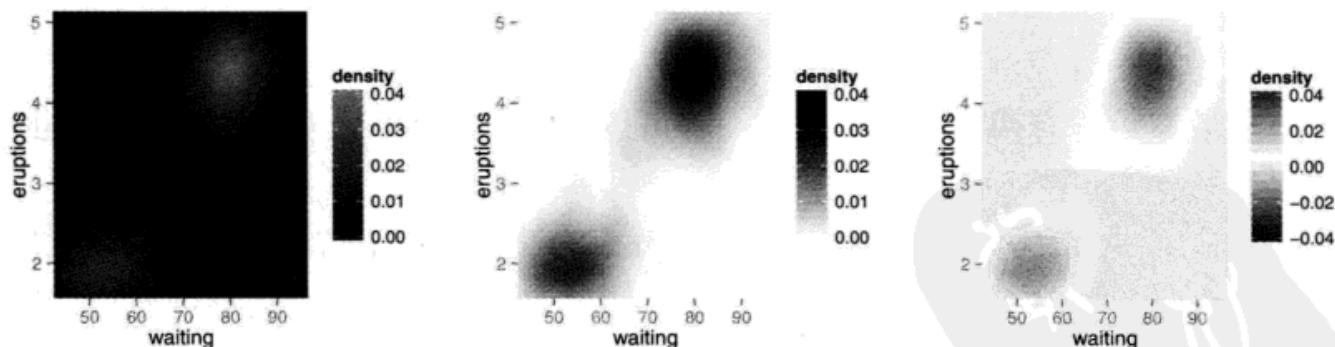


图 6.7 (参见彩图 21) 温泉喷发时间数据密度估计的三种配色方案。(左图) 默认的颜色梯度, (中图) 自定义的黑白梯度以及 (右图) 中点设为密度均值的 3 点梯度。

要创建自定义的颜色梯度, 使用 `scale_colour_gradientn()` 即可。这在颜色能够代表数据的特征时很有用 (例如, 某个黑体的颜色或标准的地形色), 或者你也可以直接使用另外一些包生成的调色板。以下代码和图 6.8 展示了使

用 vcd 包生成的调色板。Zeileis et al. (2008) 阐述了这些调色板背后的哲学，并介绍了创建一个优秀颜色标度的复杂性。

```

1 library(vcd)
2 fill_gradn <- function(pal) {
3   scale_fill_gradientn(colours = pal(7), limits = c(0, 0.04))
4 }
5 erupt + fill_gradn(rainbow_hcl)
6 erupt + fill_gradn(diverge_hcl)
7 erupt + fill_gradn(heat_hcl)

```

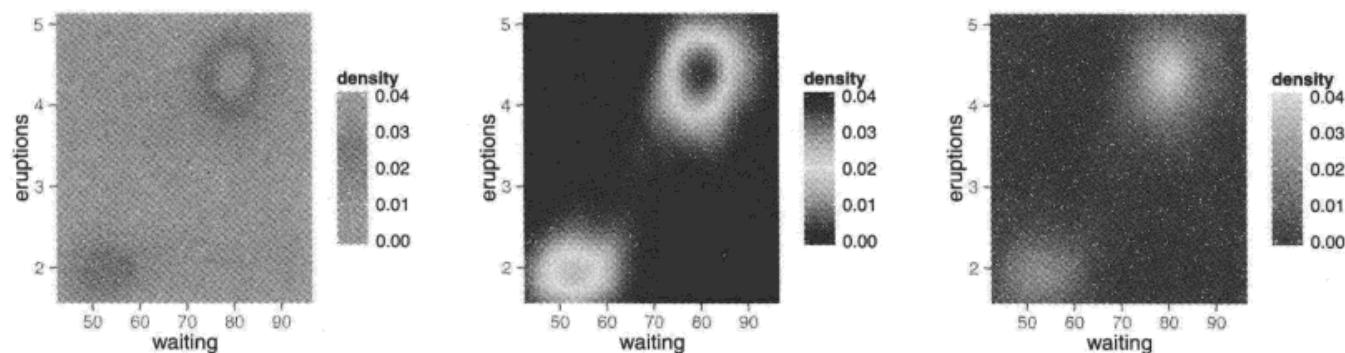


图 6.8 (参见彩图 22) 使用 vcd 包生成的可被良好感知的梯度型颜色标度。从左至右分别为：顺序型、发散型，以及 hcl 热度型调色板。每种标度使用 `scale_fill_gradientn` 生成，参数 `colours` 分别设置为 `rainbow_hcl(7)`、`diverge_hcl(7)` 和 `heat_hcl(7)`。

离散型

离散型数据有两种颜色标度。一种可以自动选择颜色，另一种可以轻松地从手工甄选的颜色集中选择颜色。

默认的配色方案，即 `scale_colour_hue()`，可通过沿着 hcl 色轮选取均匀分布的色相来生成颜色。这种方案对多至约 8 种颜色时都能有较好的效果，但对于更多的颜色，要区分开不同的颜色就变得比较困难了。默认配色的另外一个缺点是，由于所有颜色都拥有相同的明度和彩度，所以当我们进行黑白打印时，它们就会成为几近相同的灰影。

除了这种基于计算的方案以外，另一种可选的方案是使用 ColorBrewer 配色，<http://colorbrewer.org>。这些手工甄选的颜色可在很多情境下良好地运作，尽管它更专注于地图，这些颜色也因此在展示较大的面积时表现更佳。对于

类别型数据中的点而言，我们最感兴趣的调色板是“Set1”和“Dark2”，对面积而言则是“Set2”、“Pastel1”、“Pastel2”和“Accent”。使用 `RColorBrewer::display.brewer.all` 可列出所有的调色板。图 6.9 展示了应用于点和条形的三种调色板，代码如下^⑥。

```
1 point <- qplot(brainwt, bodywt, data = msleep, log = "xy",
2   colour = vore)
3 area <- qplot(log10(brainwt), data = msleep, fill = vore,
4   binwidth = 1)
5
6 point + scale_colour_brewer(palette = "Set1")
7 point + scale_colour_brewer(palette = "Set2")
8 point + scale_colour_brewer(palette = "Pastel1")
9 area + scale_fill_brewer(palette = "Set1")
10 area + scale_fill_brewer(palette = "Set2")
11 area + scale_fill_brewer(palette = "Pastel1")
```

如果要用自制的离散型颜色标度，可使用下文所述的 `scale_colour_manual()`。

6.4.4 手动离散型标度

离散型标度 `scale_linetype()`, `scale_size_discrete()` 和 `scale_shape()` 基本上没有选项（虽然对于形状标度，你可以选择点是空心可填充的或是实心的）。这些标度仅仅是按一定的顺序将因子的水平映射到一系列取值中。

如果你想要定制这些标度，你需要使用以下手动型标度创建新的标度：`scale_shape_manual()`, `scale_linetype_manual()`, `scale_colour_manual()` 等等。手动型标度拥有一个参数 `values`，你可以使用它来指定这个标度应该生成的值。如果这个向量中的元素是有名称的，则它将自动匹配输入和输出的值，否则它将按照离散型变量中水平的先后次序进行匹配。你需要了解一些可用的图形属性值，详见附录 B。以下代码^⑦演示了 `scale_manual()` 的使用方法，结果见图 6.10。

```
1 plot <- qplot(brainwt, bodywt, data = msleep, log = "xy")
2 plot + aes(colour = vore) +
```

^⑥译者注：原始代码中函数 `scale_fill_brewer()` 的参数 `pal` 在新版本中名为 `palette`。

^⑦译者注：原始代码中函数 `scale_colour_manual()` 的参数名 `value` 在新版本中为 `values`。

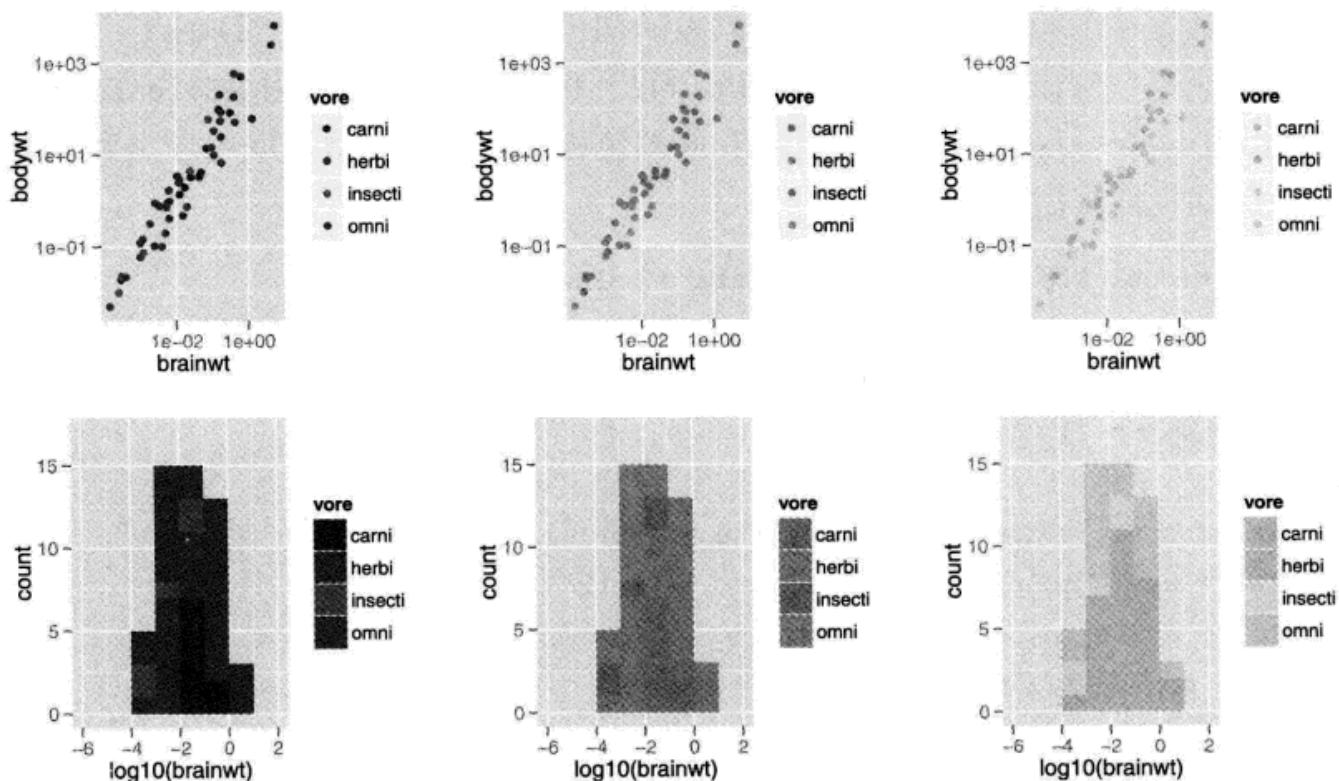


图 6.9 (参见彩图 23) 三种 ColorBrewer 调色板, Set1(左图), Set2(中图) 和 Pastel1(右图), 分别应用于点(首行)和条形(次行)。更明亮的颜色对点效果良好, 但对于条形来说过于刺眼。淡色对条形效果良好, 但会让点看不清楚。

```

3   scale_colour_manual(values = c("red", "orange", "yellow",
4     "green", "blue"))
5   colours <- c(carni = "red", NA = "orange", insecti = "yellow",
6     herbi = "green", omni = "blue")
7   plot + aes(colour = vore) + scale_colour_manual(values = colours)
8   plot + aes(shape = vore) +
9   scale_shape_manual(values = c(1, 2, 6, 0, 23))

```

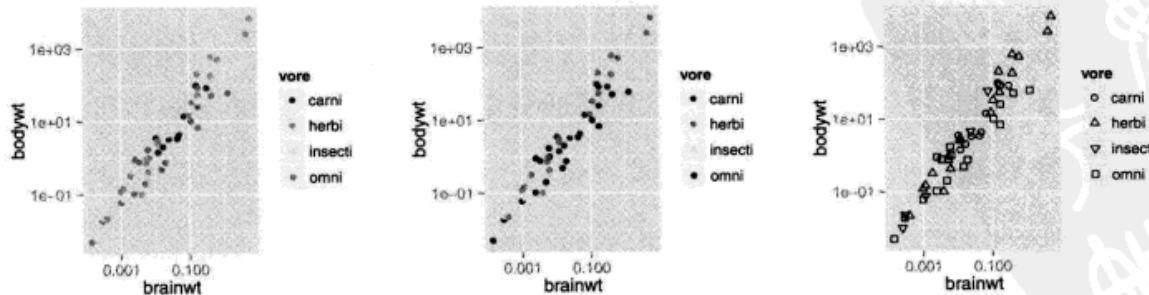
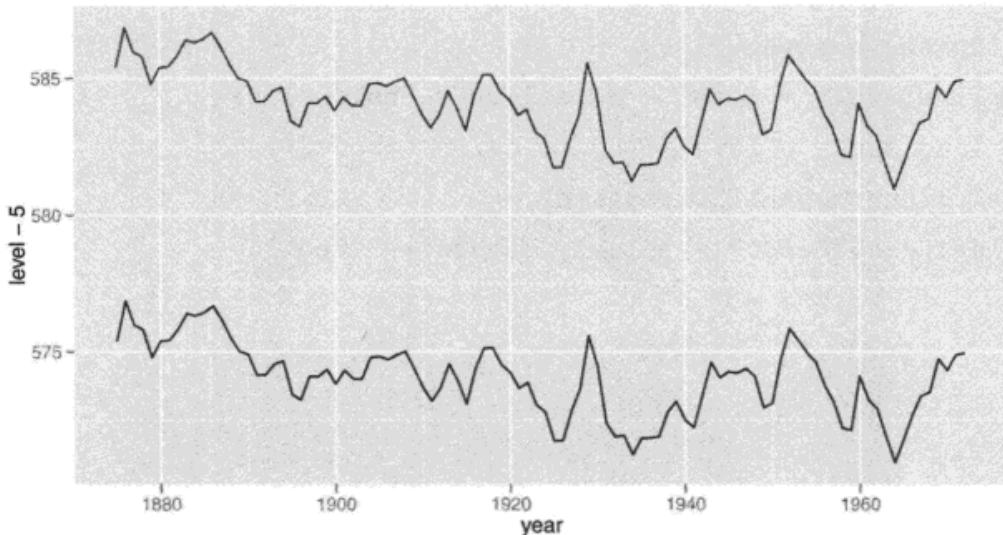


图 6.10 (参见彩图 24) 使用手动型标度创建自定义颜色标度(左图和中图)和自定义形状标度(右图)。

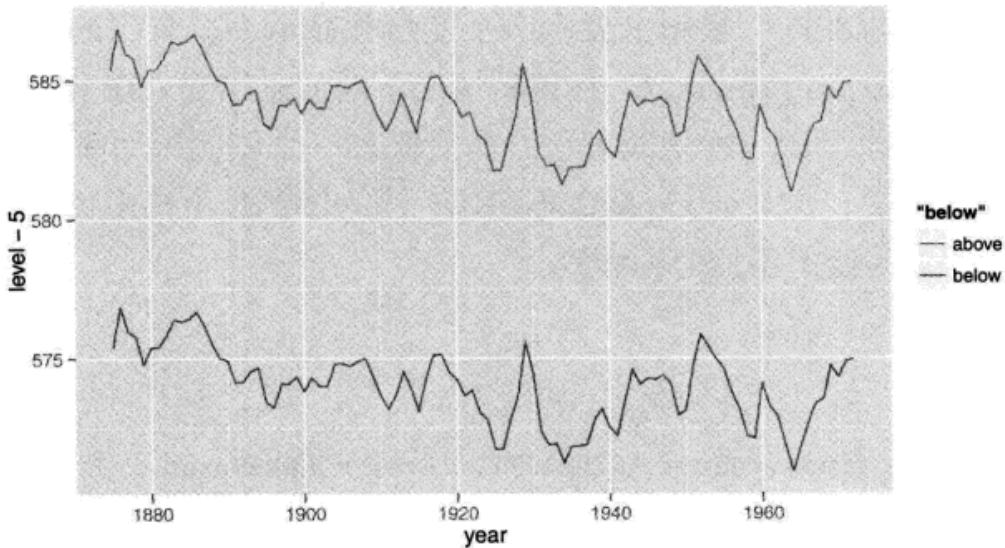
有时你可能想在同一幅图上展示多个变量并显示一个有用的图例，下例就展示了 `scale_colour_manual()` 在这种情况下的创意用法。在多数其他的图形系统中，你只需像下图一样把线上色，然后添加一个图例说明哪种颜色对应着哪个变量就可以了。这对于 `ggplot2` 不适用，因为图例是由标度负责绘制的，但标度并不知道要为线条添加何种标签。

```
1 > huron <- data.frame(year = 1875:1972, level = LakeHuron)
2 > ggplot(huron, aes(year)) +
3 +   geom_line(aes(y = level - 5), colour = "blue") +
4 +   geom_line(aes(y = level + 5), colour = "red")
```



你需要做的，就是通过创建从数据到颜色图形属性的一个映射，来告诉颜色标度这两条线的信息。此数据中并未出现变量，所以需要先创建一个：

```
1 > ggplot(huron, aes(year)) +
2 +   geom_line(aes(y = level - 5, colour = "below")) +
3 +   geom_line(aes(y = level + 5, colour = "above"))
```

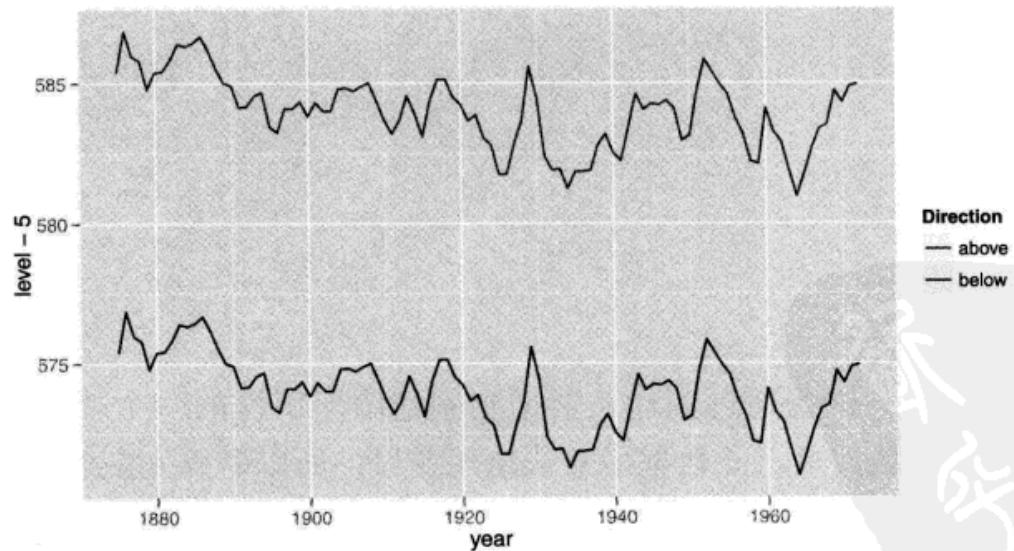


这样基本得到了我们想要的效果，但图例标注不正确且颜色有误。可以使用 `scale_colour_manual`^⑧ 进行修复：

```

1 > ggplot(huron, aes(year)) +
2   + geom_line(aes(y = level - 5, colour = "below")) +
3   + geom_line(aes(y = level + 5, colour = "above")) +
4   + scale_colour_manual("Direction",
5     +   values = c("below" = "blue", "above" = "red"))

```



此问题的另一种解决方法参见 9.2.1 节。

^⑧译者注：与原始代码不同，这里我们必须显式指定函数 `scale_colour_manual()` 参数 `values` 的名称。

6.4.5 同一型标度

当你的数据能被 R 中的绘图函数理解时，即数据空间和图形属性空间相同时，可以使用同一型标度 (identity scale)。这就意味着此时无法仅从数据本身派生出有意义的图例，所以默认是不绘制图例的。如果你确实想要绘制一个，则仍然可以通过使用参数 `breaks` 和 `labels` 自行设定。

图 6.11 展示了一类适合使用 `scale_identity` 的数据。这里数据本身即为颜色值，我们也无法创造一个有意义的图例。同一型标度在手工将数据标度为图形属性的情况下也是有用的。在那种情况下，你将不得不自己考虑哪些断点和标签对你的数据来说是有意义的。

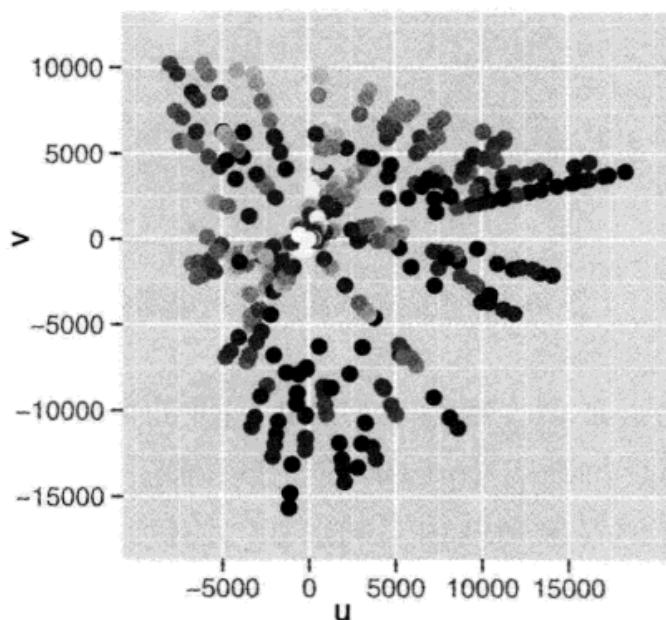


图 6.11 (参见彩图 25)R 中的颜色在 Luv 空间的图形。这里没必要添加图例，因为点的颜色就已经代表了它们自身：数据空间和图形属性空间是相同的。

6.5 图例和坐标轴

坐标轴和坐标被共同称为引导元素，它们都是标度的逆函数：它们允许你在图中读出观测并将其映射回原始值。图 6.12 标出了引导元素和它们的组成部分。图例和坐标轴存在着天然的可比性：图例标题 (legend title) 和坐标轴名 (axis label) 是等价的，并且都由标度的名称参数 (name) 决定；图例标示 (legend key) 和刻度标签 (tick label) 皆由标度的断点参数 (break) 决定。

在 `ggplot2` 中，图例和坐标轴可根据你在图形中使用的标度和几何对象自动

生成。这一点与其他图形系统中由你负责自行添加图例的机制不同。在 `ggplot2` 中，你几乎无法直接控制图例。乍一看，这似乎是一个很大的限制，但随着你逐渐适应这种方式，你将会发现这样可以节约大量的时间，并且（图例的）几乎所有方面都是可以控制的。

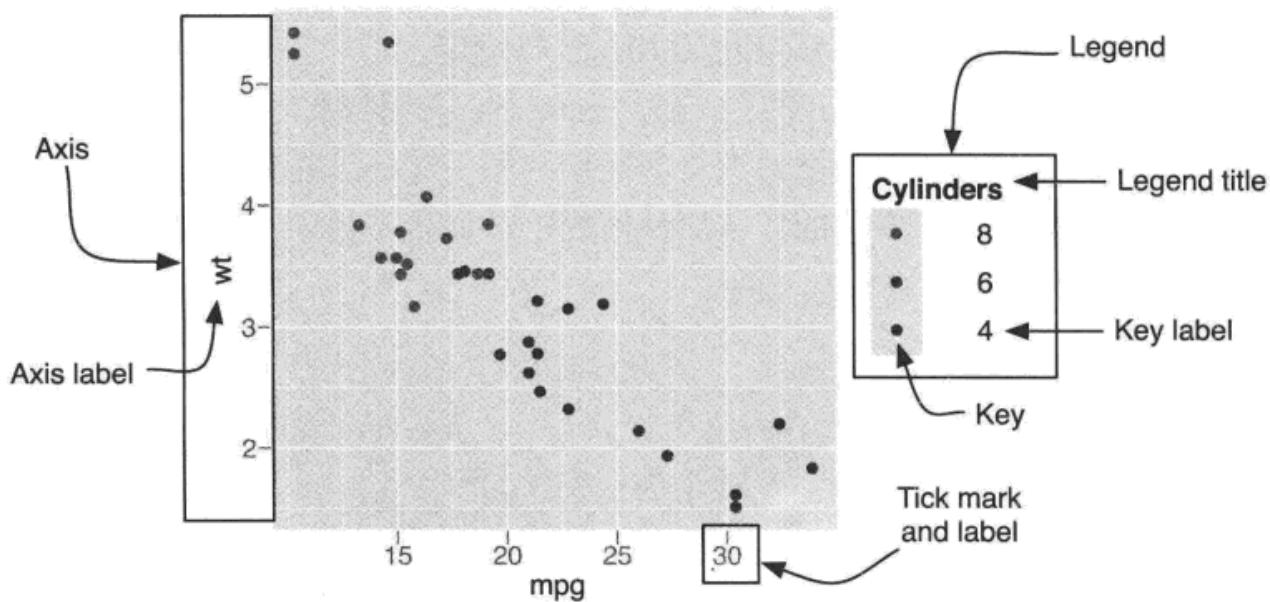


图 6.12 坐标轴和图例的组成部分。

要绘制图例，图形必须收集每一种图形属性的使用信息：为何种数据以及为何种几何对象。标度的断点（breaks）被用来确定图例标示的值，使用了对应图形属性的一系列几何对象则被用来确定如何绘制这些标示。举例来说，如果你使用了点这个几何对象，那么你将在图例中得到点；如果你使用了线，你将会在图例中得到线。如果点和线都被使用了，那么点和线都会被绘制在图例中。图 6.13 阐明了这一点。

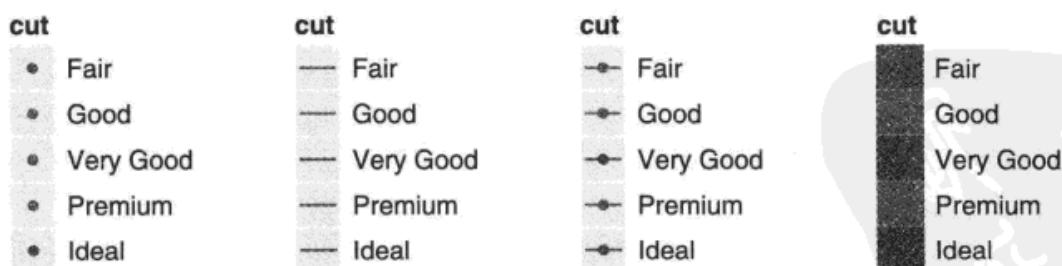


图 6.13（参见彩图 26）由几何对象生成的图例：点、线、点和线，以及条形。

`ggplot2` 会尝试生成最小数量的、能够精确表达图中使用图形属性的图例。当一个变量对应了多个图形属性时，`ggplot2` 可以通过合并图例的办法来达到精简的目的。图 6.14 使用点来展示了一个这方面的例子：如果颜色和形状都被映

射到相同的变量，那么用一个图例就足够了。被合并的图例必须拥有相同的名称（相同的图例标题）。因此，如果你修改了某个合并后的图例的名称，你将需要修改所有的相关名称。

cut	cut	cut
● Fair	● Fair	● Fair
● Good	▲ Good	▲ Good
● Very Good	■ Very Good	■ Very Good
● Premium	+ Premium	+ Premium
● Ideal	▣ Ideal	▣ Ideal

图 6.14 (参见彩图 27) 颜色图例、形状图例、颜色 + 形状图例。

图例和坐标轴的内容是由标度控制的，而渲染的细节是由主题系统控制的。以下列表包含了最常用的微调选项。

- 标度 `name` 控制着坐标轴名和图例标题，可为字符串或数学表达式，语法详见 `?plotmath` 所述。
- 本章早些时候介绍了，`breaks` 和 `labels` 是非常重要的标度函数参数，因为它们控制着哪些刻度标签出现在坐标轴上，以及哪些标示出现在图例上。如果默认选择的断点不合适（或是你想使用信息量更足的标签）时，设置这些参数即可调整图例标示和坐标刻度的外观。
- 主题设置 `axis.*` 和 `legend.*` 控制着坐标轴和图例的整体外观。要了解如何操纵这些选项，请参看 8.1 节。
- 内部网格线由主要断点和次要断点的参数控制。默认地，次要的网格线均匀地分布在原始的数据空间中：这就为对数——对数图形赋予了通用的行为，即主要网格线是可乘的，而次要网格线是可加的。你可以使用参数 `minor_breaks` 覆盖掉次要网格线。网格线的外观是使用 `panel.grid.major` 和 `panel.grid.minor` 两个主题设置来控制的。
- 图例的位置和对齐是使用主题设置 `legend.position` 来控制的，其值可为 `right`、`left`、`top`、`bottom`、`none`（不加图例），或是一个表示位置的数值。这个数值型位置由 `legend.justification` 给定的相对边角位置表示（取 0 和 1 之间的值），它是一个长度为 2 的数值型向量：右上角为 `c(1, 1)`，左下角为 `c(0, 0)`。

6.6 更多资源

在我们不断尝试选择不同的图形属性和新标度的同时，了解图形将被如何感知是非常重要的。部分优秀的参考资料如下：

- Cleveland (1993a, 1985); Cleveland and McGill (1987) 研究了图形如何被感知以及图形化编码数据的最佳方式。
- Tufte (1990, 1997, 2001, 2006) 介绍了如何绘制美观的，含有丰富数据的图形。
- Brewer (1994a, b) 介绍了如何选择在多种情境下都表现良好的配色方案，特别是对于面积图。
- Carr (1994, 2002); Carr and Sun (1999) 介绍了使用颜色的一般规律和方法。



第 7 章 定位

7.1 简介

本章的主题是图形的定位，主要讲解了布局分面和坐标系如何工作两个问题。定位由四个部分组成，在之前与分面有关的篇章中，我们已经学习了其中的两个部分：

- **位置调整** 调整每个图层中出现重叠的对象的位置，在 4.8 节有过介绍。该方法对条形图和其他有组距的几何对象非常有用，同时也可以用于其他的情形。
- **位置标度** 控制数据到图形中位置的映射，在 6.4.2 小节有过介绍。常用的映射是线性和对数变换；当然，你也可以使用其他可逆的变换。

本章将对其余两部分进行讲解，同时也将展示如何对这四个组成部分进行综合应用：

- **分面** 即在一个页面上自动摆放多幅图形的技法：先将数据划分为多个子集，然后将每个子集依次绘制到页面的不同面板中。这类图形也通常被称作小联号图 (small multiples)，可参见本章 7.2 节。
- **坐标系** 即通过控制两个独立的位置标度来生成一个 2 维的坐标系。最常见的笛卡尔坐标系，在某些特定的情形中也可以使用其他种类的坐标系。

7.2 分面

在 2.6 节介绍函数 `qplot()` 时，你其实已经接触了分面的用法：切割数据生成一系列小联号图，每个小图表示不同的数据子集。对探索性数据分析来说，

分面是一个强大有力的工具。它能帮你快速地分析出数据各子集模式的异同。本节将讨论如何较好地微调分面，特别是与位置标度相关的方法。

`ggplot2` 提供两种分面类型：网格型 (`facet_grid`) 和封装型 (`facet_wrap`)。网格分面生成的是一个 2 维的面板网格，面板的行与列通过变量来定义；封装分面则先生成一个 1 维的面板条块，然后再封装到 2 维中。网格分面布局类似于 R 基础图形中的 `coplot` 布局，而封装分面则类似于 `lattice` 的面板布局。由图 7.1 我们可以清楚地看到两者的不同之处。

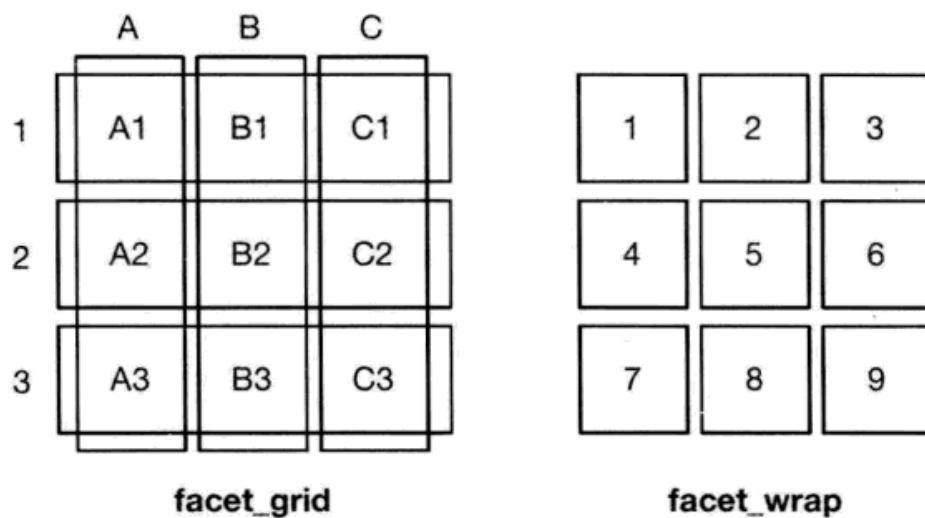


图 7.1 两种分面系统间差异的示意图。`facet_grid()` (左) 本质上为 2 维，由两个独立的部分组成；`facet_wrap()` (右) 本质上为 1 维，为节省空间而封装成 2 维。

分面系统有两个基本参数：一个是分面变量的设置，另一个是指定分面的位置标度是全局还是局部。两套分面系统在参数设定上略有不同，在后面的章节里我们将对两者分别进行讲解。

在 `qplot()` 中可以选择分面系统。2 维分面 (比如 `x~y`) 使用 `facet_grid`，1 维分面 (比如 `~x`) 使用 `facet_wrap`。

分面绘图通常会占用大量空间，因此本节使用 `mpg` 数据集的子集来进行展示。它有几个简单的水平：三种气缸 (4, 6, 8) 和两种驱动轮 (4 和 f)。子集剔除了原数据中的 29 辆汽车。

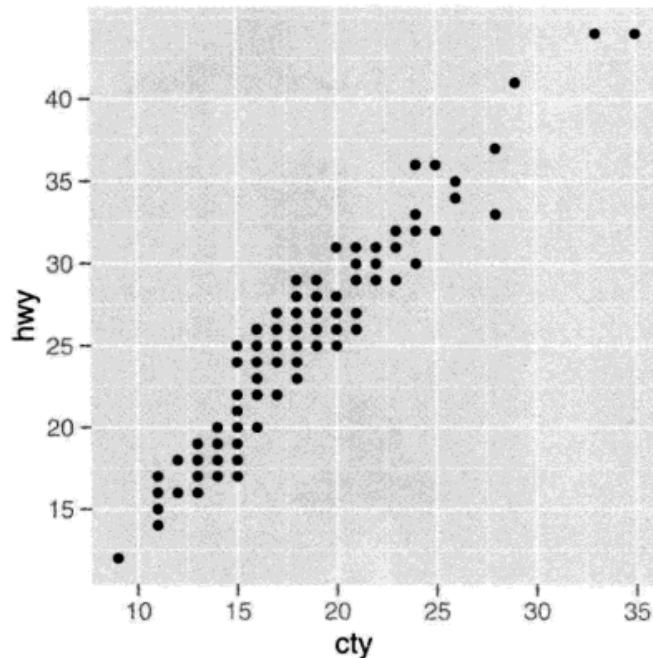
```
1 mpg2 <- subset(mpg, cyl != 5 & drv %in% c("4", "f"))
```

7.2.1 网格分面

网格分面在 2 维网格中展示图形。输入分面表达式时，你需要设定哪些变量作为分面绘图的行，哪些变量作为列，规则如下：

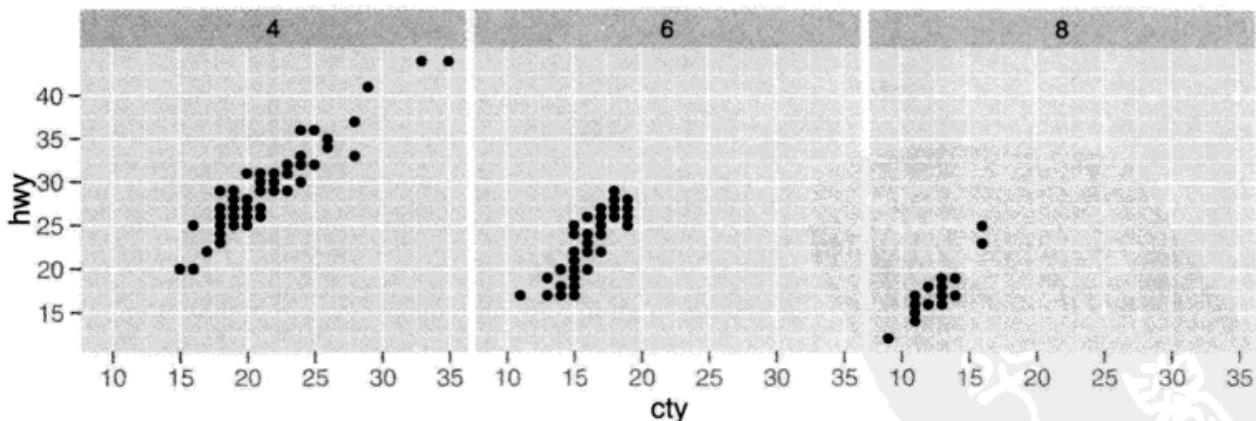
- 不进行分面：即不使用函数 `facet_grid` 或加上命令 `facet_null()`，此时，没有行或列被分面化，我们将得到一个单独的面板^①。

```
1 ## 和 qplot(cty, hwy, data = mpg2) 效果一样
2 qplot(cty, hwy, data = mpg2) + facet_null()
```



- 一行多列："`. ~ a`"。电脑屏幕通常较宽，因此这个方向最适合数据的展示。另外，因为纵坐标轴相同，这个方向也有助于 y 位置的比较。

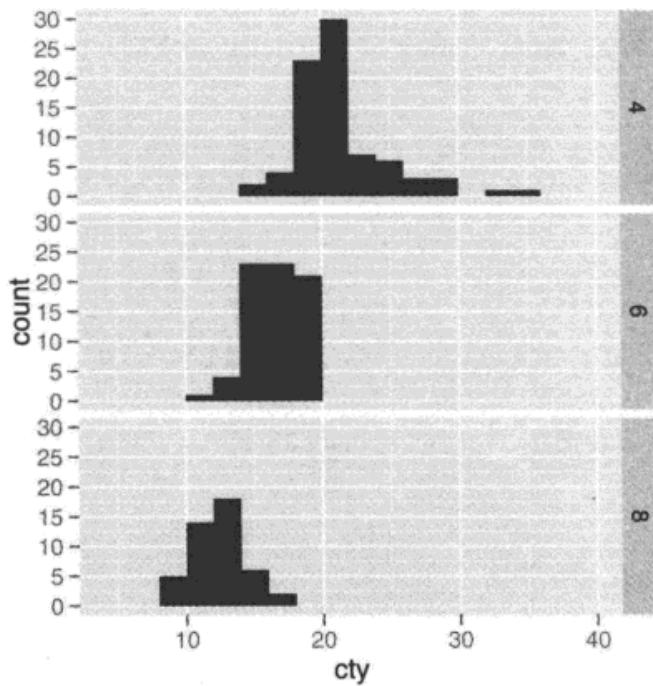
```
1 qplot(cty, hwy, data = mpg2) + facet_grid(. ~ cyl)
```



- 一列多行："`b ~ .`"。横坐标轴相同，利于 x 位置的比较，尤其是对数据分布的比较。

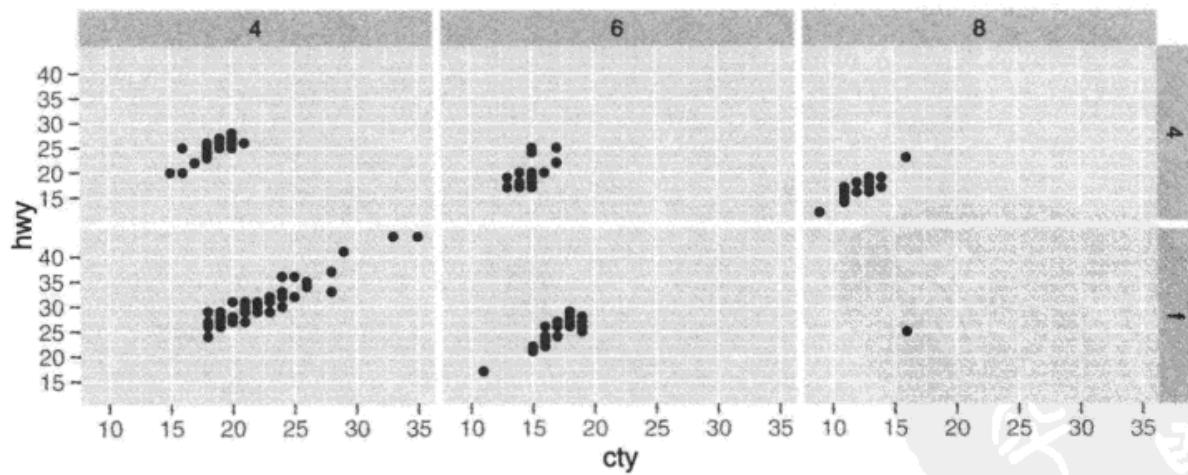
^①译者注：0.9.0 之前版本中 `facet_grid(. ~ .)` 表示不分面。

```
1 qplot(cty, data = mpg2, geom="histogram", binwidth = 2) +
2   facet_grid(cyl ~ .)
```



- 多行多列: "a ~ b"。我们通常都将因子水平数目最大的变量按列排放,这样可以充分利用屏幕的宽高比。

```
1 qplot(cty, hwy, data = mpg2) + facet_grid(drv ~ cyl)
```



- 多个变量的多个水平在行或者列上 (或者同时): . ~ a + b 或 a + b ~ ..一般来说此法不太实用,除非变量的因子水平个数比较少,或电脑屏幕很宽,或需要生成一个瘦长的海报的时候。

从以上图形我们可以看到，变量在某行或某列一起出现时，图形只会展示数据中出现的变量组合。而变量在行和列都出现时，图形将展示所有的变量组合，包括在原始数据中都没有出现的组合，不过这也可能会导致出现空白面板。

边际图

切割图形就好比创建一个列联表。列联表可展示每个单元格的值以及边际和（一个行或列的总和）。这种模式对于绘图也非常有用，你可以使用参数 `margins` 来绘制边际图。

设定 `margins = TRUE` 可展示所有的边际图，或者如 `margins = c("sex", "age")`，列出你要展示的边际图的变量名称。另外，你也可以使用 `grand_row` 或 `grand_col` 来分别生成所有行或所有列的边际图。

如图 7.2，左图没有边际图，右图展示了所有的边际图。列边际图和行边际图分别展示了所有的驱动轮和所有的汽缸，右下角的分面则展示了整个数据集的情况。从图中我们可以看到，随着气缸数增加，发动机排量将增加，燃油经济性将下降；相比前轮驱动的汽车，四轮驱动汽车有大致相同的排量，但燃油经济性较低。图形生成代码如下：

```
1 p <- qplot(displ, hwy, data = mpg2) +  
2   geom_smooth(method = "lm", se = F)  
3 p + facet_grid(cyl ~ drv)  
4 p + facet_grid(cyl ~ drv, margins = T)
```

边际图的分组方式与其他面板中分组方式相同：默认使用图层中所有分类变量的交互作用（可以参见 4.5.3 小节）。下面的代码表示对每个驱动轮类型添加了彩色平滑线。

```
1 qplot(displ, hwy, data = mpg2) +  
2   geom_smooth(aes(colour = drv), method = "lm", se = F) +  
3   facet_grid(cyl ~ drv, margins = T)
```

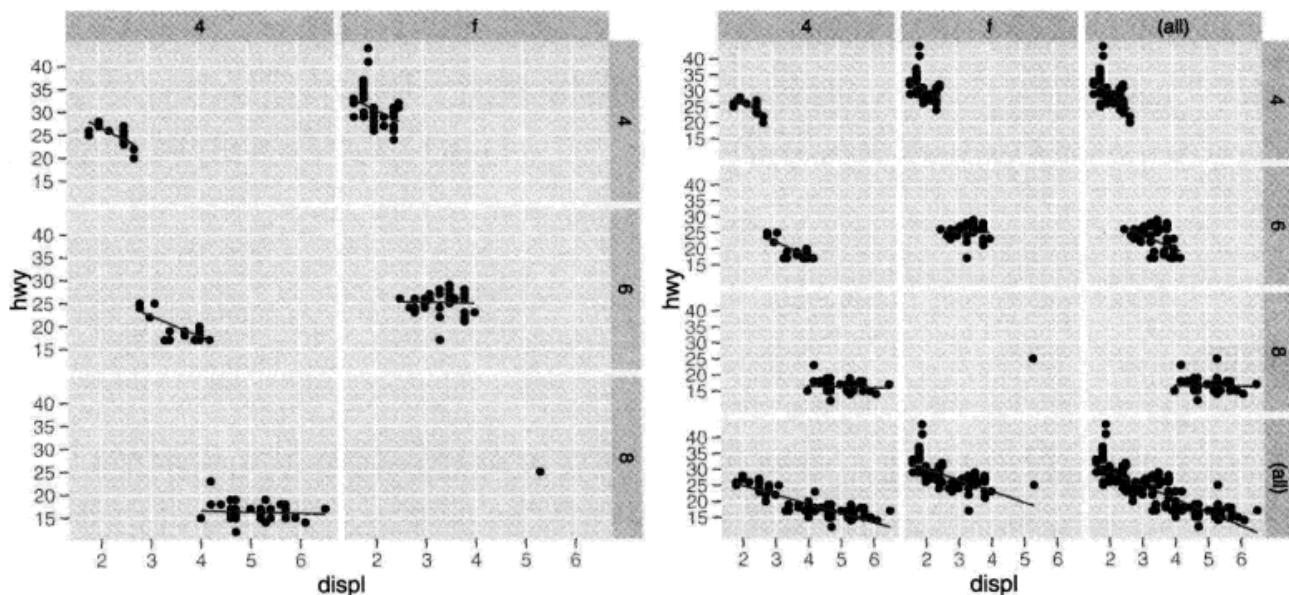
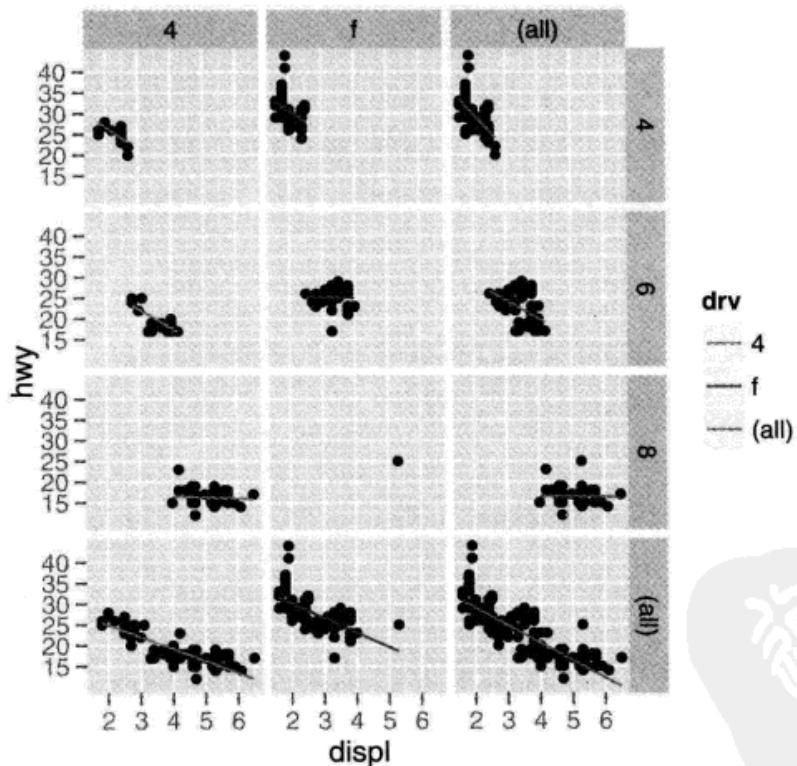


图 7.2 图形的边际图生成与列联表类似，提供给我们一个无依赖条件的数据观察视角。左图展示了根据气缸数和驱动器数生成的分面图形，右图补充了它们的边际图。



相对于屏幕展示，具有许多分面和边际图的图形更适合印刷，因为印刷有更高的分辨率 (600dpi vs 72dpi)，这样你可对更多的数据子集进行比较。

7.2.2 封装分面

另一种分面形式是封装式的图形条块。`facet_wrap` 并不是用两个或者更多的变量来生成一个 2 维网格，而是先生成一个长的面板条块（由任意数目的变量生成），然后将它封装在 2 维中。在处理单个多水平变量时，这种处理方式非常有用，它可以有效地利用空间来安放图形。`lattice` 中栅栏（Trellis）图形也采取了这个方法。

图 7.3 展示了每十年电影平均评分的分布情况。主要差异看起来似乎是随时间变化，评分分布趋于平坦。原因很可能是投票中的人为选择因素：越新的电影将会得到更多的投票，因此平均打分不会显得太极端。这种类型分面图的缺点是：由于分面利用条块式封装，所以在对比某些原本相邻的子集时会很困难。比如图中 50 年代与 60 年代评分面板本应该离的很近，但现在却相离很远，不利于比较。图形代码如下：

```

1 movies$decade <- round_any(movies$year, 10, floor)
2 ## round_any() 需要加载 plyr 包
3 qplot(rating, ..density.., data = subset(movies, decade > 1890),
4       geom = "histogram", binwidth = 0.5) +
5       facet_wrap(~ decade, ncol = 6)

```

分面变量的设置形式为 `~ a + b + c`。`facet_wrap` 默认把图形面板尽可能摆放成方形，且更偏好于宽扁型的矩形。你可以通过设置 `ncol`, `nrow` 来更新默认设置，更多实例请参考文档。

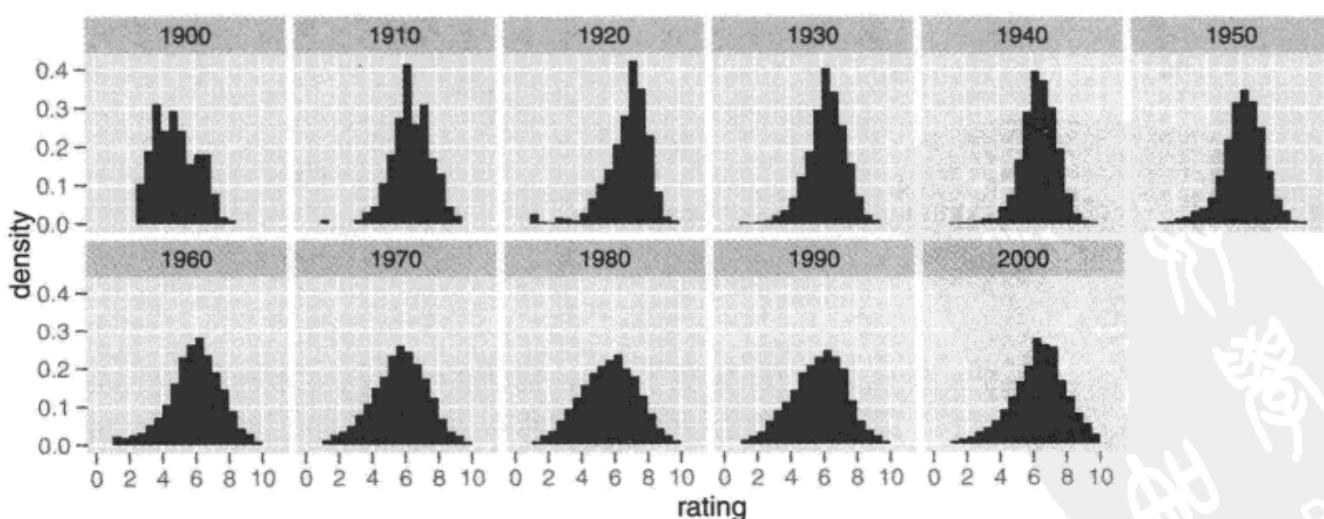


图 7.3 每十年电影平均评分的分布。

7.2.3 标度控制

对于两种分面，你可以通过调整参数 `scales` 来控制面板的位置标度是相同（固定）还是允许变化（自由）：

- `scales = "fixed"`: x 和 y 的标度在所有面板中都相同。
- `scales = "free"`: x 和 y 的标度在每个面板都可以变化。
- `scales = "free_x"`: x 的标度可变，y 的尺度固定。
- `scales = "free_y"`: y 的标度可变，x 的尺度固定。

图 7.4 展示了固定标度与自由标度间的差异。

```
1 p <- qplot(cty, hwy, data = mpg)
2 p + facet_wrap(~ cyl)
3 p + facet_wrap(~ cyl, scales = "free")
```

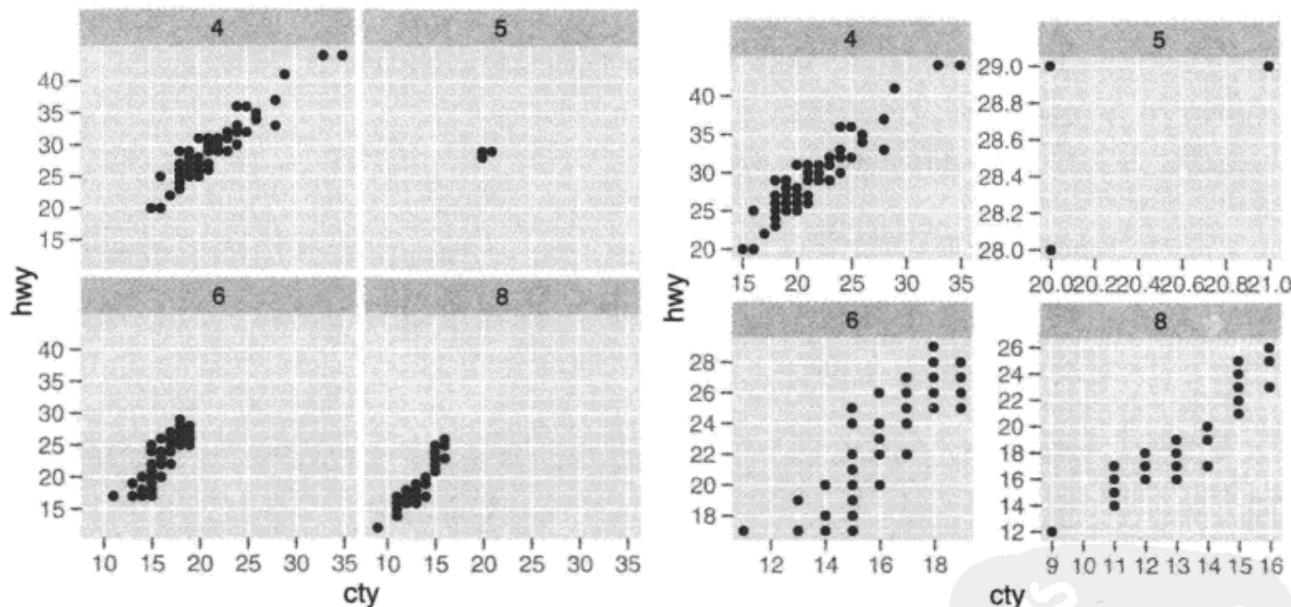


图 7.4 每个分面中固定标度即横纵坐标范围相同（左图），而自由标度即横纵坐标范围可变（右图）。

固定标度可以让我们在相同的基准上对子集进行比较，观察在哪些地方各个子集有相似的总体模式。而自由标度可以帮助我们发现更多细节，它在展示不同量纲的时间序列时非常有用。为展示自由标度的效果，我们首先需要把数据从“宽”变“长”，把不同的变量堆叠在一列中。图 7.5 为我们展示了一个示例，更多的细节在 9.2 小节将作更多的探讨。

```

1 em <- melt(economics, id = "date")
2 ## 0.9.0 之后版本中 melt() 函数需要加载 reshape2 包
3 library(reshape2)
4 qplot(date, value, data = em, geom = "line", group = variable) +
5   facet_grid(variable ~ ., scale = "free_y")

```

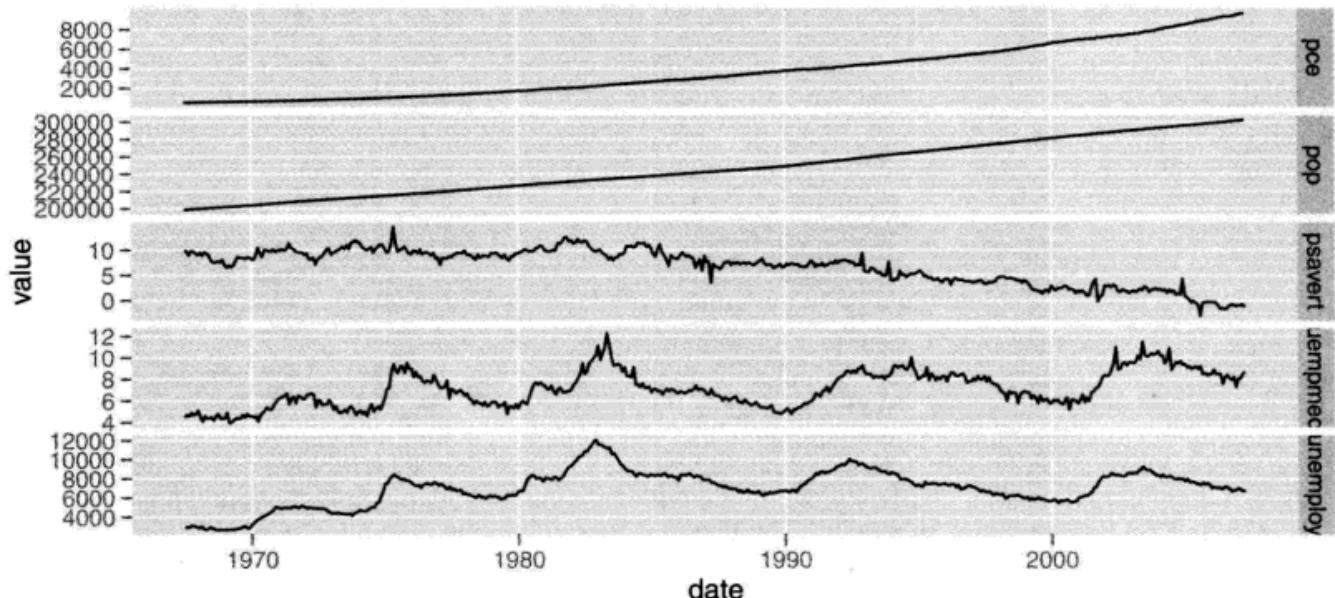


图 7.5 自由标度在展示不同量纲的时间序列时非常有用。

使用网格分面时 `facet_grid` 有一个额外的限制：同列的面板必须有相同的 x 标度，同行的面板必须有相同的 y 标度。这是因为网格分面中，每列都共用一个 x 轴，每行都共用一个 y 轴。

`facet_grid` 还有一个额外的参数 `space`，值可为 "free" 或 "fixed"^②。当 `space` 设定为 `free` 时，每列（行）的宽度（高度）与该列（行）的标度范围成比例。这将使得所有面板的标度比例相同：每个面板中的 1cm 都映射为相同的数据范围（与 `lattice` 中的“sliced”轴的限制类似）。例如，若面板 a 有 2 个单位的范围，面板 b 有 4 个单位的范围，那么三份之一的空间将分配给 a，其余的分配给 b。这对分类标度非常有用，图 7.6 给出了具体的实例，代码如下。代码中的 `reorder()` 可使模型 (model) 和生厂商 (manufacturer) 按城市油耗 (cty) 重新排序。

```

1 mpg3 <- within(mpg2, {
2   model <- reorder(model, cty)

```

^②译者注：0.9.0 及之后版本中，`space` 多增加了两个参数 `free_x` 和 `free_y`。

```

3   manufacturer <- reorder(manufacturer, -cty)
4 }
5 models <- qplot(cty, model, data = mpg3)
6
7 models
8 models + facet_grid(manufacturer ~ ., scales = "free",
9   space = "free") + theme(strip.text.y = element_text())

```

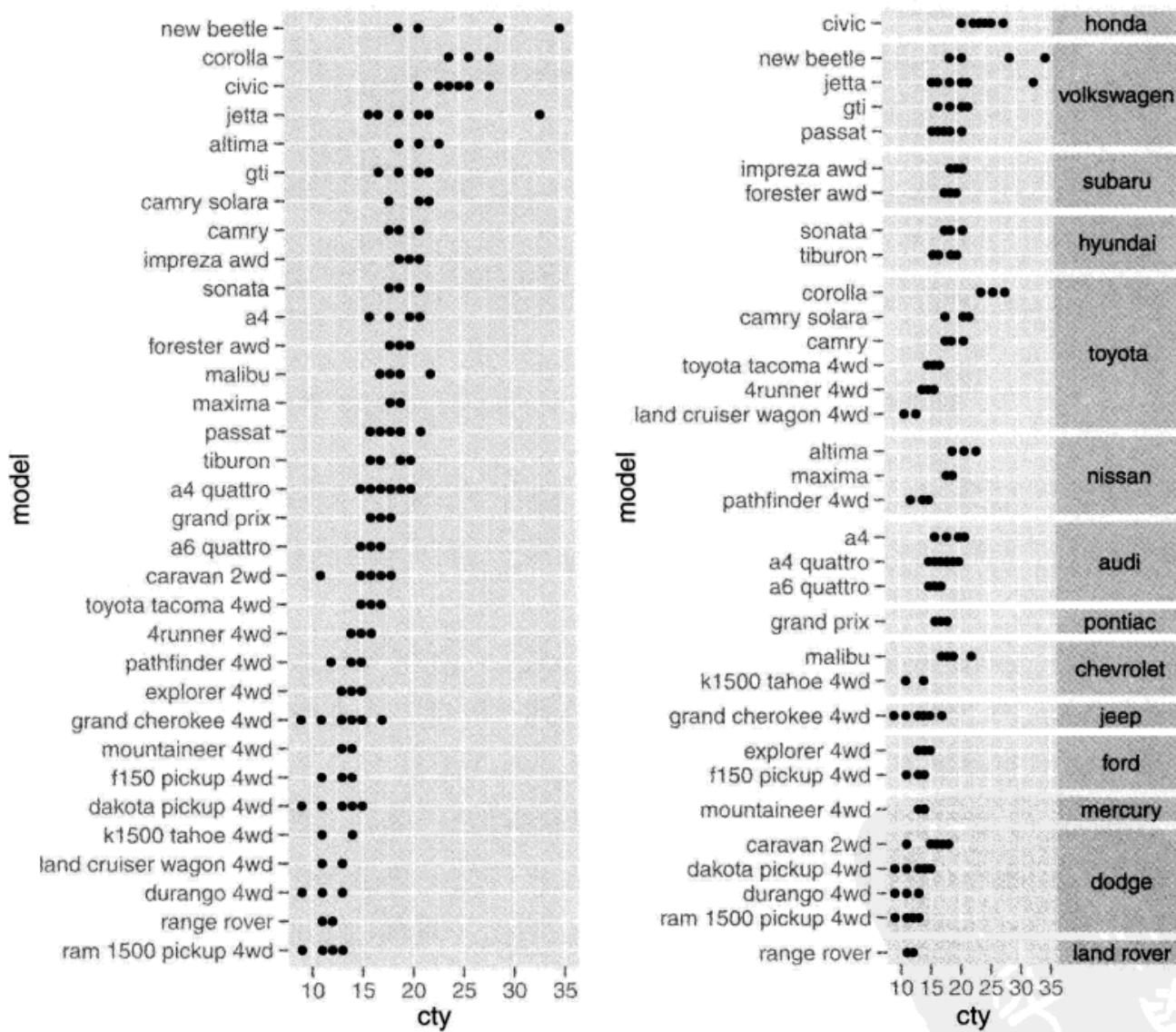


图 7.6 每种小汽车的城市油耗英里数的点图。(左) 车类型按 mpg 均值排序, (右) 使用 `scales = "free_y"` 和 `space = "free"`, 按生产商进行分面。`strip.text.y` 主题设置用来旋转分面标签。

7.2.4 分面变量缺失

假如我们想对一个含有多个数据集的图形使用分面，而其中一个数据集缺失了分面变量，那么图形将会出现什么情况呢？在对所有图形面板添加相同背景信息时，你会经常遇到这种问题。例如，你想对疾病的空间分布按性别进行分面绘图，但一个新添加的地图图层不包含性别变量，这时如何处理呢？`ggplot2`会按照你所期望的那样，给每个面板都添加地图，此时缺失的分面变量按包含该分面变量所有的值来处理。

7.2.5 分组与分面

与通过调整图形属性（比如颜色、形状或大小）来分组不同，分面提供了另外一种分组途径。依据子集相对位置的不同，这两种绘图技巧都有相应的优缺点。

在分面图形中，每个组别都在单独的面板中，相隔较远，组间无重叠。因此组与组之间重叠严重时，分面图形有一定的好处，不过这也会导致组间的细微差别难以发现。使用图形属性区分各组时，各组将会离得很近甚至可能重叠，不过细微的差别容易被发现。图 7.7 展示了两者优缺点的互补：在散点图中，仅使用颜色区分各组，我们可能还没意识到图形重叠了，但是回归线却可以让我们看到 D、E 和 G 被分组在一起，与 J 组相距较远。图形代码如下：

```
1 xmaj <- c(0.3, 0.5, 1, 3, 5)
2 xmin <- as.vector(outer(1:10, 10^c(-1, 0)))
3 ymaj <- c(500, 1000, 5000, 10000)
4 ymin <- as.vector(outer(1:10, 10^c(2, 3, 4)))
5 dplot <- ggplot(subset(diamonds, color %in% c("D", "E", "G", "J")),
6   aes(carat, price, colour = color)) +
7   scale_x_log10(breaks = xmaj, labels = xmaj, minor = xmin) +
8   scale_y_log10(breaks = ymaj, labels = ymaj, minor = ymin) +
9   scale_colour_hue(limits = levels(diamonds$color)) +
10  theme(legend.position = "none")
11
12 dplot + geom_point()
13 dplot + geom_point() + facet_grid(. ~ color)
14
15 dplot + geom_smooth(method = lm, se = F, fullrange = T)
```

```

16 dplot + geom_smooth(method = lm, se = F, fullrange = T) +
17   facet_grid(. ~ color)

```

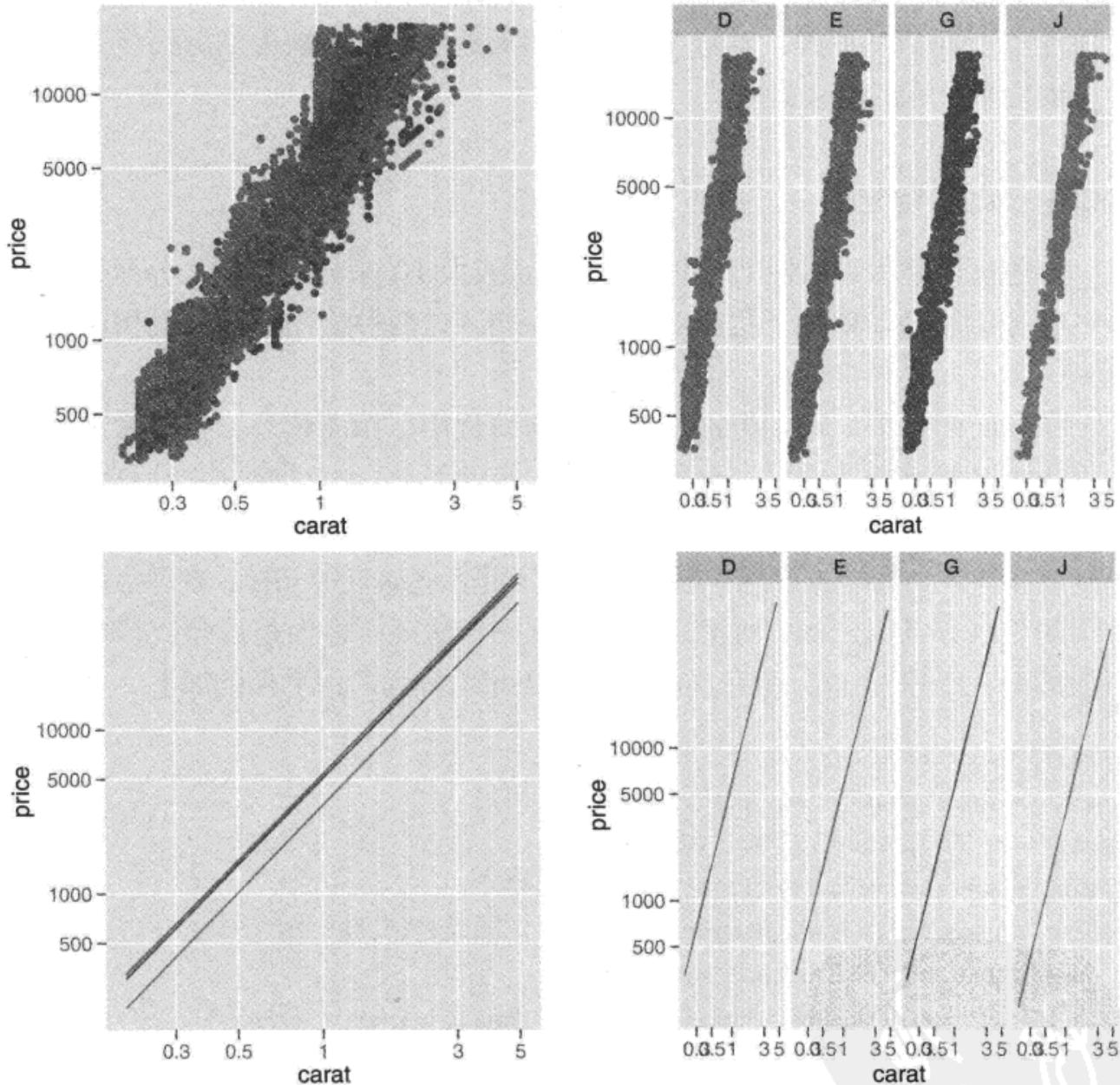


图 7.7 (参见彩图 28) 分面与分组的差异：克拉与价格的双对数图，四种颜色标注。

当组别数很多时，分面也能较好处理。因为你可将它们划分到 2 个维度中，这时比较两个变量相对于比较两个不同图形属性会更为容易。分面还有其他的优势，比如数据子集若有不同的尺度范围，各个面板中标度可以作相应调整。

7.2.6 并列与分面

分面可绘制出与图形并列类似的图形效果。图 7.8 中，并列和分面绘制的图形看起来非常相似，它们主要的区别在于标注方式：分面图形在上方有颜色的标注，同时下面有切工的标注，而并列图形在下方有颜色标注，切工却没有清晰标注出来。在本例中，分面图形中的标签需要做一些调整，以增强可读性，具体细节参照如下代码：

```

1 qplot(color, data = diamonds, geom = "bar", fill = cut,
2       position = "dodge")
3 qplot(cut, data = diamonds, geom = "bar", fill = cut) +
4   facet_grid(. ~ color) +
5   theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 8,
6                                     colour = "grey50"))

```

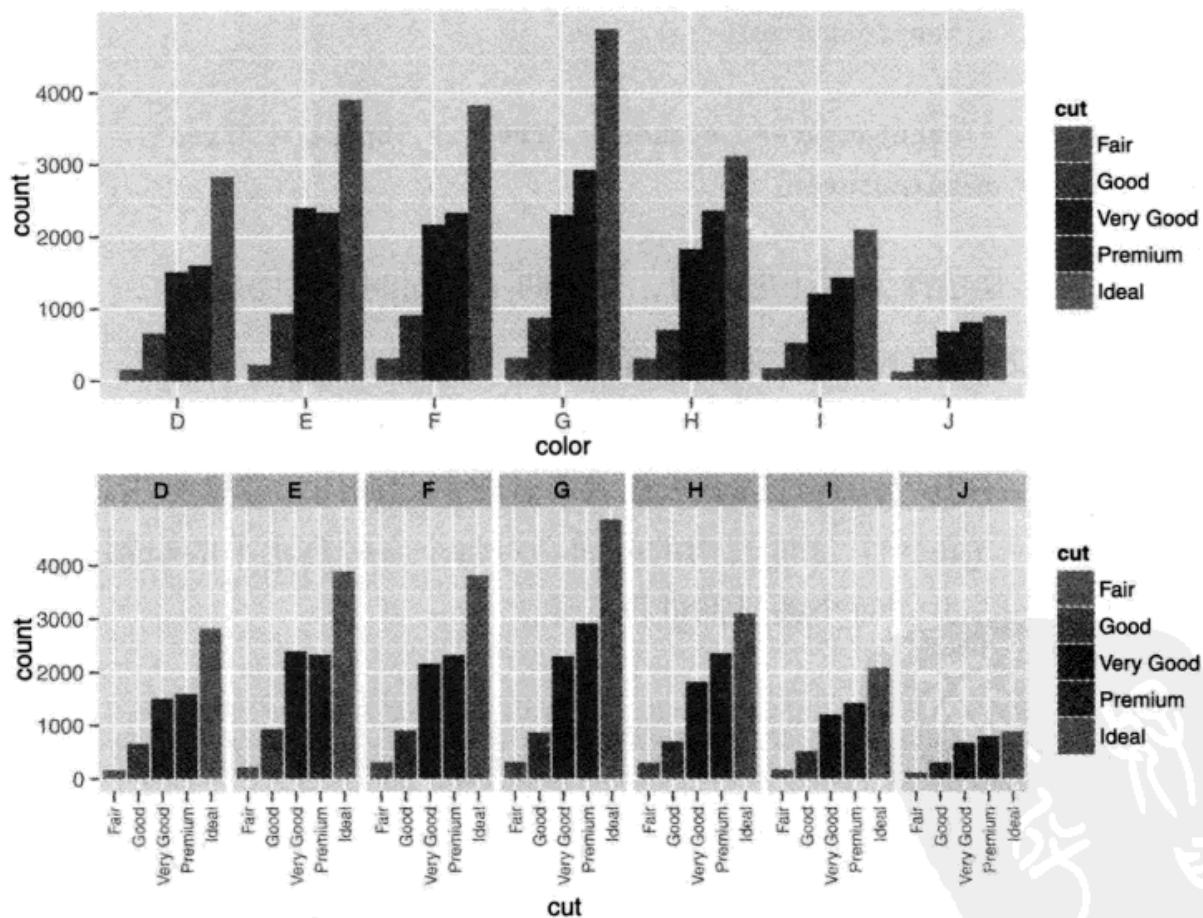


图 7.8 (参见彩图 29) 并列 (上) vs 分面 (下) 对于水平完全交叉的变量对。

除标注方式外，当两个变量的因子水平几乎完全交叉，而部分变量组合缺失时，两种绘图方式也就会有所不同。此时，并列图形的用处不大，因为它只

是对条形图做局部地分割，没有任何标签。而分面就实用得多，它能控制分割方式是局部的（`scales = "free_x"`, `space = "free"`）还是全局的（`scales = "fixed"`）。图 7.9 对两者效果进行了比较，数据来源于 `mpg` 数据集，变量 `model` 和 `manufacturer` 为嵌套类型，代码如下：

```

1 mpg4 <- subset(mpg, manufacturer %in%
2   c("audi", "volkswagen", "jeep"))
3 mpg4$manufacturer <- as.character(mpg4$manufacturer)
4 mpg4$model <- as.character(mpg4$model)
5
6 base <- ggplot(mpg4, aes(fill = model)) +
7   geom_bar(position = "dodge") +
8   theme(legend.position = "none")
9
10 base + aes(x = model) +
11   facet_grid(. ~ manufacturer)
12 last_plot() +
13   facet_grid(. ~ manufacturer, scales = "free_x", space = "free")
14 base + aes(x = manufacturer)

```

总之，图形是选择分面还是并列，要视两变量间的关系而定：

- 水平完全交叉：分面和并列基本等同。
- 水平几乎交叉：有相同标度的分面保证了所有的水平组合可见，即使有些是空的。当存在非结构性的缺失组合时，绘制分面图形非常有用。
- 水平无交叉（嵌套）：标度自由的分面会对每个有较高水平的组别分配充足的作图空间，并对每个条目都进行标注。

7.2.7 连续型变量

对连续型变量进行分面，首先需要将其变换为离散型。有三种转化方法：

- 将数据分为 n 个长度相同的部分：用 `cut_interval(x, n = 10)` 控制划分数目，或用 `cut_interval(x, length = 1)` 控制每个部分的长度。控制划分数目会容易些，但划分区间的端点值可能不太“美观”。
- 将数据划分为 n 个有相同数目点的部分：`cut_number(x, n = 10)`。这使

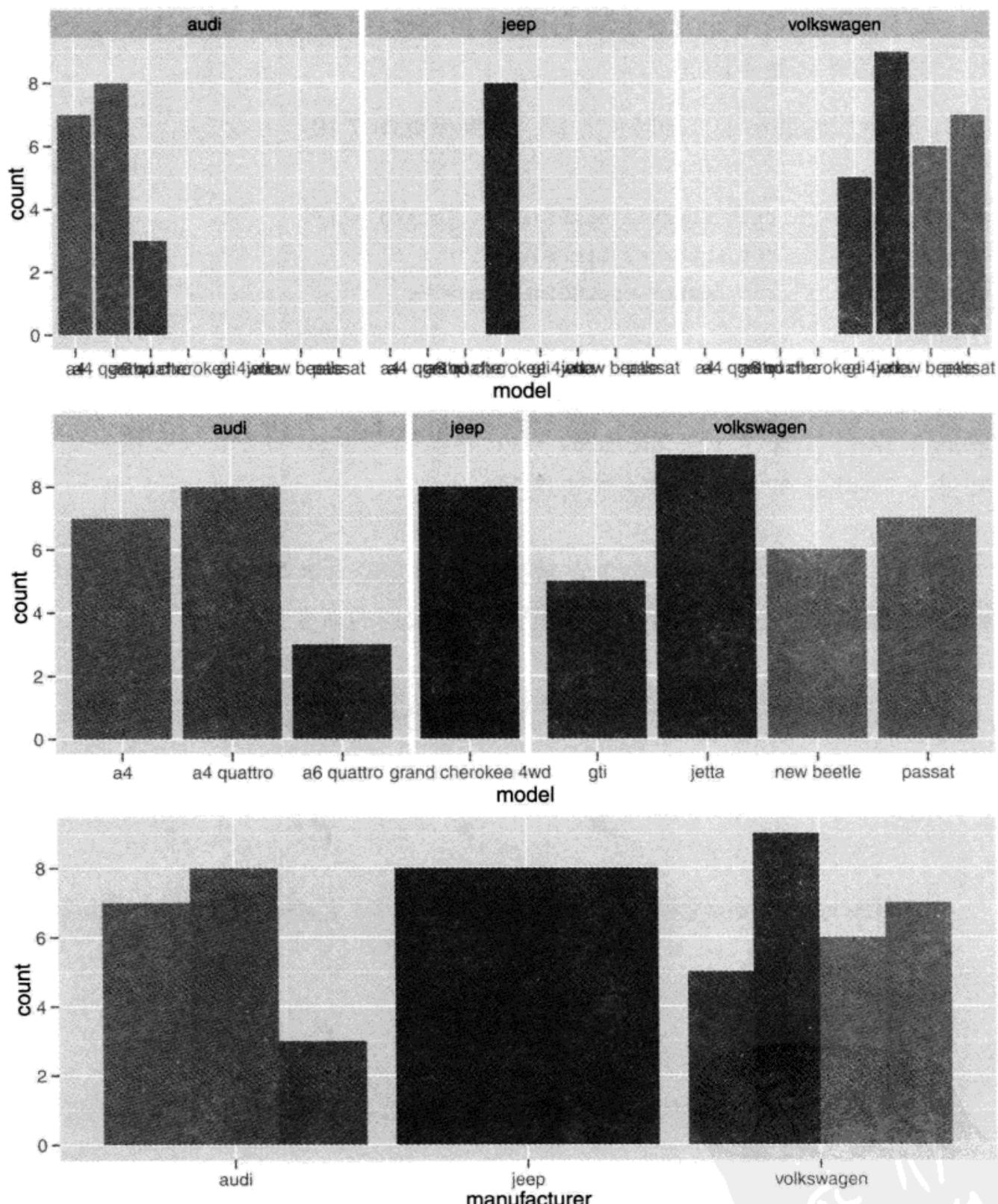


图 7.9 (参见彩图 30) 对于嵌套型数据, 分面(上, 中) 的优势相比并列(下)是明显的, 它能控制图形并进行标注。此例中, 上图不是很实用, 但对于几乎交叉的数据非常有用, 比如缺失了一个水平组合的情况。

得分面间进行对比会更容易（分面有相同数目的点），但需要注意每个部分的标度范围是不同的。

下面的代码展示了三种转化方式，结果参见图 7.10。

```

1 mpg2$disp_ww <- cut_interval(mpg2$displ, length = 1)
2 mpg2$disp_wn <- cut_interval(mpg2$displ, n = 6)
3 mpg2$disp_nn <- cut_number(mpg2$displ, n = 6)
4
5 plot <- qplot(cty, hwy, data = mpg2) + labs(x = NULL, y = NULL)
6 plot + facet_wrap(~ disp_ww, nrow = 1)
7 plot + facet_wrap(~ disp_wn, nrow = 1)
8 plot + facet_wrap(~ disp_nn, nrow = 1)

```

注意：分面表达式只对包含在数据集中的变量有效果（不是变量的函数），因此需在数据集中创建一个包含了离散化后的数据的新变量。

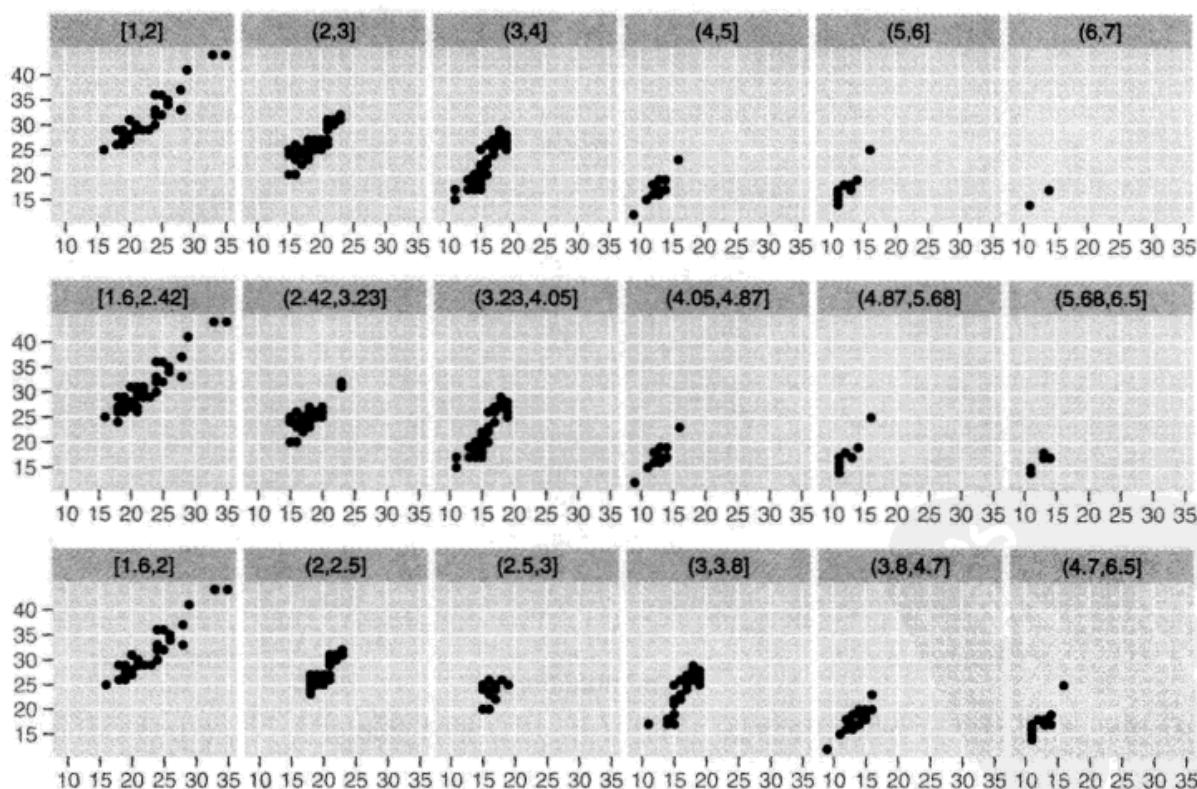


图 7.10 将连续变量离散化的三种方式。从上至下：面板长度为 1；每个面板等长；每个面板包含数目的点相同。

7.3 坐标系

坐标系是将两种位置标度结合在一起组成的 2 维定位系统。ggplot2 包含了 6 种不同的坐标系，见表 7.1。表中所有的坐标系都是 2 维的，不过我希望将来也能把 3 维图形添加进来。与 ggplot2 其他组成部分命名类似，坐标系命名规则为 `coord_` 加上坐标系的名字。比如大部分图形都默认为笛卡尔坐标系，即常见的 x 和 y 坐标组成的 2 维坐标系，命名为 `coord_cartesian()`。

坐标系主要有两大功能：

- 将 2 个位置图形属性组合起来在图形中形成 2 维方位系统。位置图形属性分别被称为 x 和 y，但将它们称作位置 1 和位置 2 可能会更合适，因为图形位置属性的名字会随着坐标系的不同而不同。比如极坐标系中将其称作角度和半径，而地图中则称作纬度和经度。
- 配合分面，坐标系将绘出坐标轴和面板背景。标度控制着坐标轴上出现的数值，并将数据映射到图形中的位置，然后通过坐标系将它们绘制出来。图形的外观随着坐标系的变化而变化，毕竟角度轴与 x 轴看起来是不同的。

表 7.1 ggplot2 中可用的坐标系。`coord_equal`、`coord_flip` 和 `coord_trans` 本质都是笛卡尔型的坐标系（维度正交），`coord_map` 和 `coord_polar` 则更为复杂。

名字	描述
cartesian	笛卡尔坐标系
equal	同尺度笛卡尔坐标系
flip	翻转的笛卡尔坐标系
trans	变换的笛卡尔坐标系
map	地图射影
polar	极坐标系

7.3.1 变换

与数据变换和标度变换不同，坐标系变换将改变图形的几何形状：在极坐标系中，矩形变为圆环的一部分；在地图中，两点间的最短路径将不是直线。图 7.11 展示了在不同的坐标系中直线和矩形的变化情况。

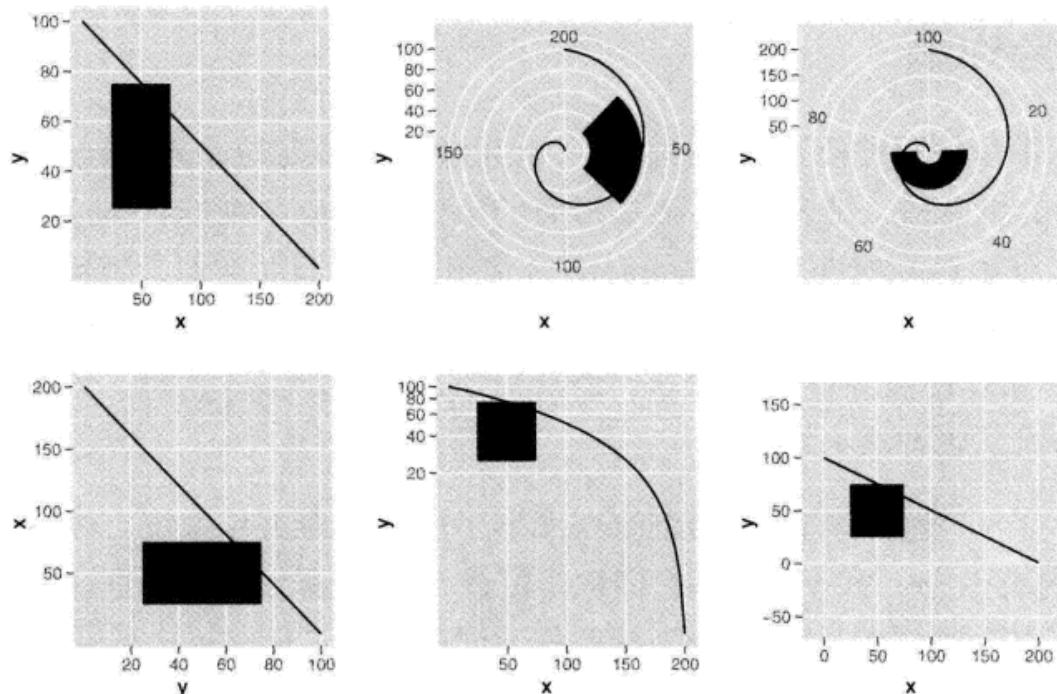


图 7.11 图解直线与矩形在不同坐标系中的变换。从左上至右下：笛卡尔坐标系，将 x 映射为角度的极坐标系，将 y 映射为角度的极坐标系，翻转的笛卡尔坐标系，y 对数变换后的笛卡尔坐标系，等标度坐标系。

坐标系变换分为两步。首先，几何形状的参数变化只依据定位，而不是定位和维度。例如，条形图由 x 位置（定位）、高度和宽度（两个维度）组成。但在一个非笛卡尔坐标系中，矩形则可能没有恒定的高度和宽度，那么如何去解析高度和宽度呢？解决方案就是仅使用基于定位的方式来表示图形。获得矩形四个角的定位后，变换位置，然后将矩形转化成多边形。如此我们可以有效地把所有几何形状转化成点、线和多边形的组合。

将几何形状变为基于定位的表现形式后，下一步就是将每个位置转化到新的坐标系中。对于点的转化，由于点在任何坐标系中都是一个点，因此它的转化相对简单，但对于线和多边形就困难得多，这是因为直线在一个新的坐标系中并不一定是直线。为解决这个问题，我们先假定坐标系之间的变换是连续的，即所有的极短的直线变换后还是很短的直线，这样，我们就可以将线和多边形切割为许多小的线段后再进行变换。这个过程被称为“分割再组合”（marching），具体可参见图 7.12：先通过两个端点使线条参数化，然后将线切割为多个很小的直线，每个都有两个端点，再将这些点转化到新的坐标系中并重新连接。在此例中，为便于看明白工作原理，分割的数目比较少，实际操作中，分割的数目非常多，这样才会使得转化结果看起来更光滑。

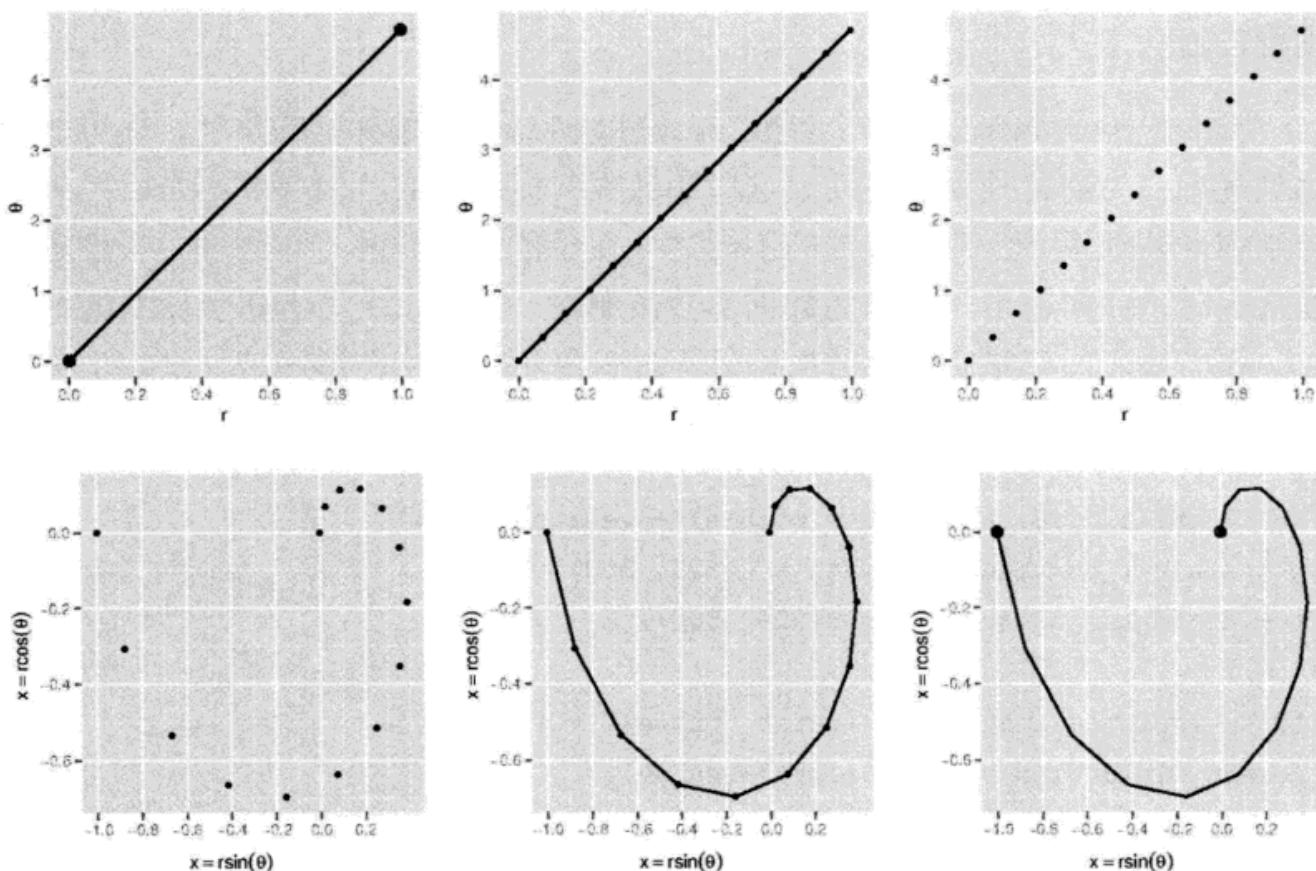


图 7.12 坐标变换；笛卡尔坐标系中的直线转化为极坐标中的曲线。初始的 x 位置转化为半径， y 位置转化为角度。

7.3.2 统计量

准确来说，原来统计变换 (stat) 使用的统计方法都依赖坐标系的选择。例如，在极坐标中，平滑曲线应使用环形回归，在 3 维中应返回一个 2 维平面而不是 1 维曲线。但是，许多统计方法都不是源自于非笛卡尔坐标系的，因此 ggplot2 仍是在笛卡尔坐标系中进行计算，这样虽然不够严格，但得出的结果一般也非常接近真实情况。

7.3.3 笛卡尔坐标系

有四种基于笛卡尔的坐标系：coord_cartesian, coord_equal^③, coord_flip 和 coord_trans，由于 x 和 y 的位置都是正交映射到图形的位置上，因此四种坐标系本质上仍是笛卡尔型的，它们之间有许多共同之处。

^③译者注：函数 coord_equal 与 coord_fixed 等价。

设置范围

`coord_cartesian` 有两个参数 `xlim` 和 `ylim`。回顾标度的章节，你可能会感到疑惑：为何此处我们还需要这些参数呢？标度中的范围参数不是已经可以控制图形外观了么？两者关键的不同之处在于范围参数的工作原理：当设定标度范围时，任何超出此范围的数据都会被删除；但当设定笛卡尔坐标系的范围时，我们使用的仍是所有的数据，只不过只展示一小片图形区域。设定坐标系范围就好比用放大镜观看图形一样。图 7.13 和图 7.14 展示了一个实例。

```

1 (p <- qplot(disp, wt, data = mtcars) + geom_smooth())
2 p + scale_x_continuous(limits = c(325, 500))
3 p + coord_cartesian(xlim = c(325, 500))
4
5 (d <- ggplot(diamonds, aes(carat, price)) +
6   stat_bin2d(bins = 25, colour="grey70") +
7   theme(legend.position = "none"))
8 d + scale_x_continuous(limits = c(0, 2))
9 d + coord_cartesian(xlim = c(0, 2))

```

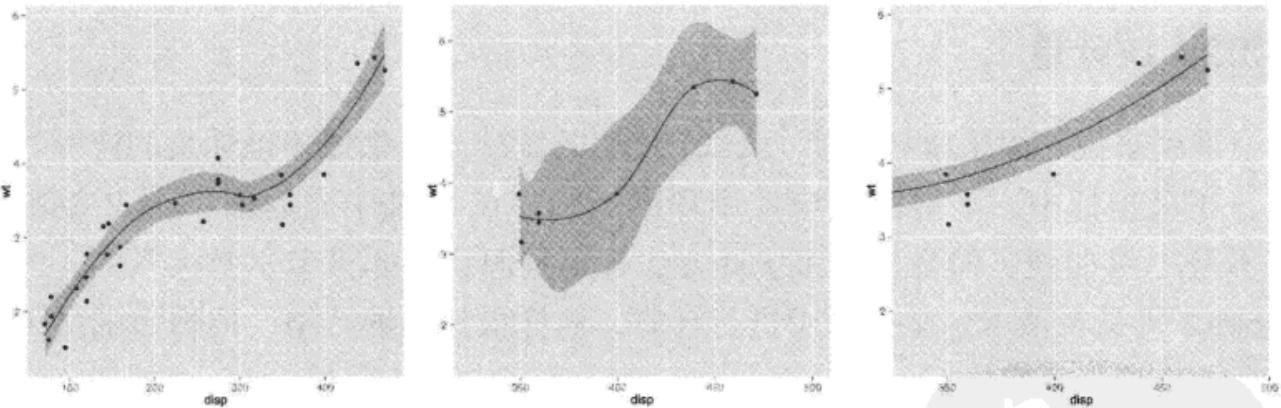


图 7.13 坐标系的范围设置 vs 标度的范围设置。(左) 完整的数据集；(中) x 的标度范围设置为 (325, 500)；(右) 坐标系 x 轴范围设置为 (325, 500)。坐标系的放缩就是图像的放缩，而标度的范围设置则是对数据取子集，然后再重新拟合曲线。

坐标轴翻转

大多数统计量和几何形状都假定你只对 x 条件下的 y 值感兴趣（比如，`smooth`, `summary`, `boxplot`, `line`），即大多数统计模型都假定 x 的值测量无误

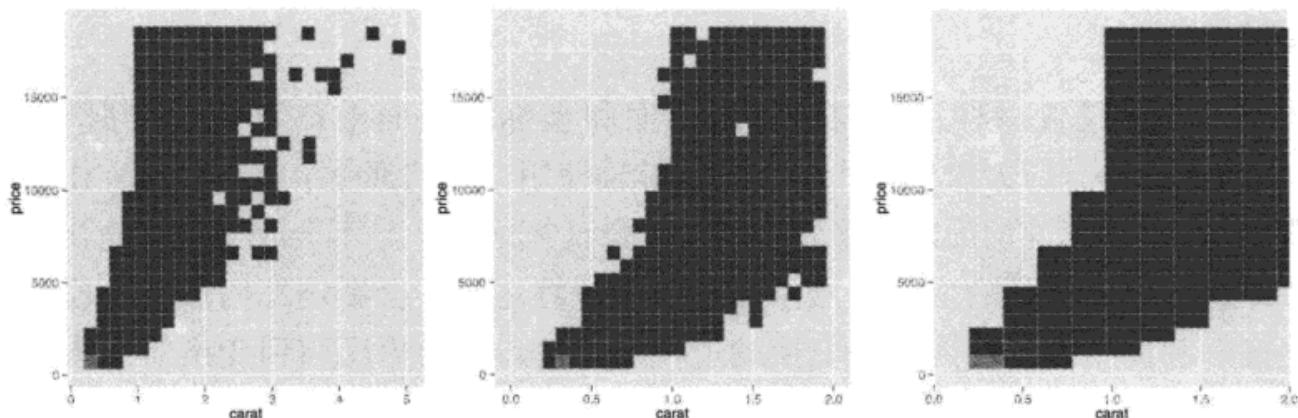


图 7.14 坐标系的范围设置 vs 标度的范围设置。(左) 完整的数据集; (中)x 的标度范围设置为 (0, 2); (右) 坐标系的 x 轴设置为 (0, 2)。比较方块的大小: 当设定标度范围时, 方块的数目还是相同的, 只是覆盖了更少数据的区域; 当设定坐标系范围时, 方块数目变少, 但他们覆盖的区域没变。

差。假如你也对 y 值条件下的 x 值感兴趣 (或仅仅是想将图形翻转 90 度), 我们便可使用 `coord_flip` 调换 x 和 y 轴。图 7.15 对交换两变量在代码中的位置与翻转坐标轴这两种方式进行了比较。

```
1 qplot(displ, cty, data = mpg) + geom_smooth()
2 qplot(cty, displ, data = mpg) + geom_smooth()
3 qplot(cty, displ, data = mpg) + geom_smooth() + coord_flip()
```

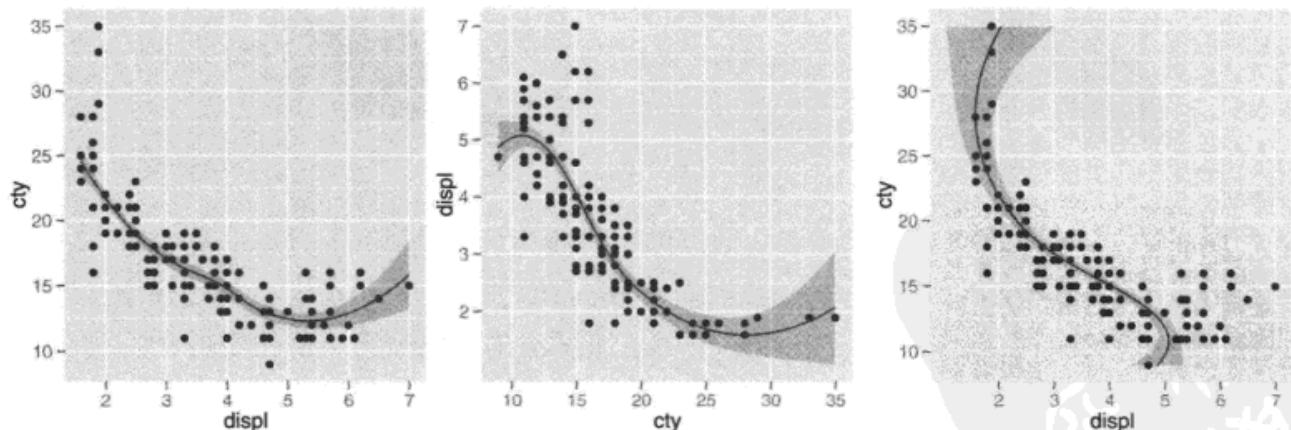


图 7.15 (左) 散点图和相应的平滑曲线, x 轴代表发动机排量 (displ), y 代表城市油耗 (cty)。(中) 互换两个变量 `cty` 和 `displ` 使图形旋转 90 度, 平滑曲线拟合的是旋转后的变量。(右) `coord_flip` 拟合初始数据, 然后再翻转输出结果, 就变成了以 y 为条件变量刻画 x 的曲线了。

变换

与范围设置一样，我们在标度层面和坐标系层面上都可进行数据变换。`coord_trans` 有 `x` 和 `y` 两个参数供坐标轴使用，它们都是字符串，被称作变换器（表 6.2）。标度层面的变换发生在统计量计算之前，且不会改变对象的几何形状。但坐标系层面的变换却发生在统计量计算之后，会影响几何形状。若两种变换一起使用，我们可先在变换的尺度上建模，然后再反演到变换前的图形以便于解释，这是常见的分析模式。图 7.16 给出了一例。

```

1 qplot(carat, price, data = diamonds, log = "xy") +
2   geom_smooth(method = "lm")
3 ## 译者注：原书中变换为"pow10"，0.9.0 之后版本为 exp_trans(10) 且需加载 scales 包
4 library(scales)
5 last_plot() + coord_trans(x = exp_trans(10), y = exp_trans(10))

```

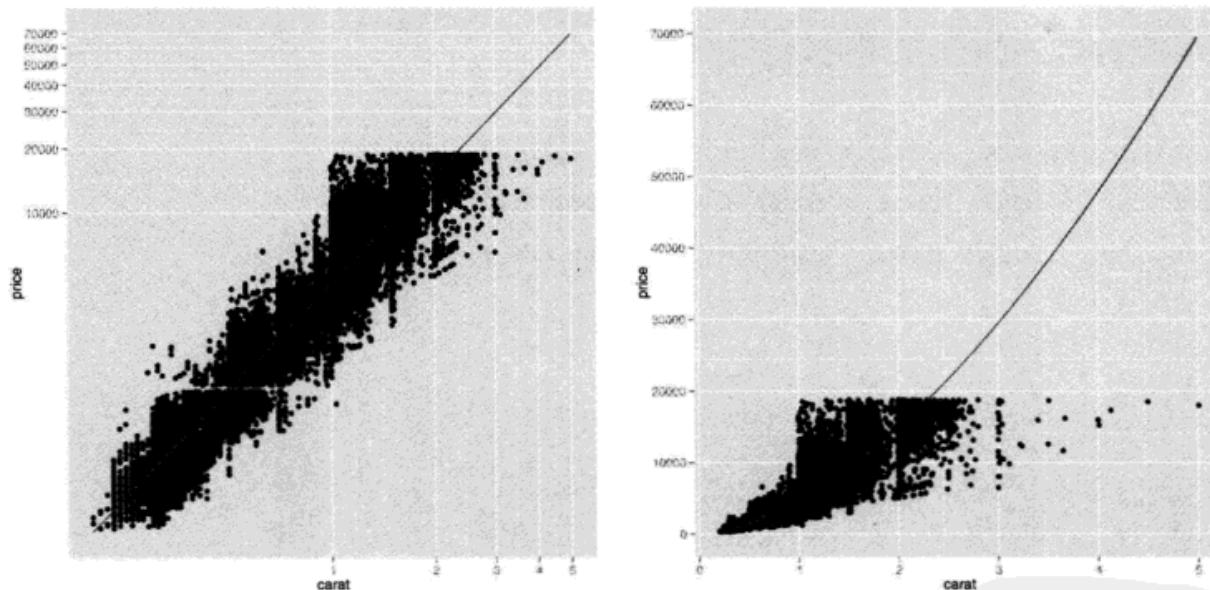


图 7.16 (左) 克拉与价格对数变换后的散点图。直线为回归曲线： $\log(y) = a + b \times \log(x)$ 。(右) 将前面的图变换回去 (`coord_trans(x = "pow10", y = "pow10")`)，标度还原，因此线性趋势变成指数形式， $y = k \times c^x$ 。图形很明显地揭示了克拉值大、价格贵的钻石是很稀少的。

相同标度

`coord_equal` 保证了 `x` 轴和 `y` 轴有相同的标度：`x` 轴上和 `y` 轴上的 1cm 代表相同的数据波动范围，默认值设定的是 1:1，我们可以通过修改参数 `ratio` 来

更改两者的尺度比例。横宽比的设定保证了不管图形输出设备的形状如何变化，映射都保持不变，更多细节可参考 `coord_equal()` 的文档。

7.3.4 非笛卡尔坐标系

非笛卡尔坐标系有两种：极坐标和地图投影。这些坐标系仍处于试验阶段，对坐标轴的布局标准比较少，因此你可以通过使用附录 C 的工具来对图形进行微调，以满足你的需要。

极坐标

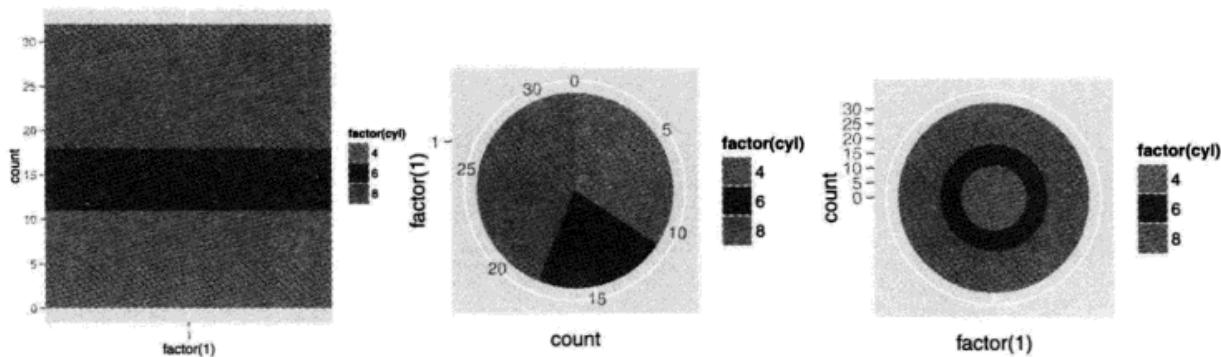


图 7.17 (参见彩图 31) (左) 堆叠的条形图。(中) 极坐标中堆叠的条形图，常被称为饼图，其中，`x` 被映射成了半径，`y` 被映射成了角度，`coord_polar(theta = "y")`。(右) 映射的变量相反，`coord_polar(theta = "x")`，这个图常被称为牛眼图。

利用极坐标可生成饼图、玫瑰图（源自条状图）和雷达图（源自直线几何对象）。但由于角度在小的半径中比在大的半径中更难被感知，因此极坐标的视觉感官性并不太佳。极坐标常被用于环形数据，特别是时间和方向数据。参数 `theta` 决定了哪个变量被映射为角度（默认为 `x`），哪个被映射为半径。图 7.17 展示了通过改变坐标系，将条状图转变成饼图和牛眼图的过程，更多实例可参考函数文档。

```

1 ## 堆叠条状图
2 (pie <- ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
3   geom_bar(width = 1))
4 ## 饼图
5 pie + coord_polar(theta = "y")
6

```

```
7 ## 牛眼图  
8 pie + coord_polar()
```

地图投影

地图仍处于试验阶段，依赖于 `mapproj` 包 (McIlroy,2005)。`coord_map()` 控制投影的参数与 `mapproj()` 相同。参考 `coord_map()` 文档可获得更多实例，当然你也可咨询专业制图师，寻求适合数据的制图方式。



第 8 章 精雕细琢

在本章中，你将学习如何绘制高质量的图形。本章重点内容是 `ggplot2` 的主题设置，通过它你可对数据之外的图形外观进行控制。另外，你也会学习到如何调整几何对象、统计变换、标度的默认值，以及与其他软件相关的图形储存方式。结合下一章的内容，利用 `grid` 图形渲染，你将学会如何控制图形视觉的方方面面，最终获得预期的、理想的图形。

图形的视觉呈现由数据和非数据部分决定。8.1 节介绍了主题系统，它控制着所有非数据展示功能。到目前为止，你应该熟悉了如何修改图形中与数据有关的部分——图层和标度——来对数据进行可视化并改变图形外观。在 8.2 节中，通过学习如何改变参数默认值，你可避免重复输入相同的参数。

8.3 节将讨论如何将 R 中的图形转化到 L^AT_EX、Word 以及其他展示或者文字处理软件中去。最后，8.4 节介绍了如何将多幅图形显示在单页面中。

8.1 主题

主题系统控制着图形中的非数据元素外观，它不会影响几何对象和标度等数据元素。主题不能改变图形的感官性质，但它可以使图形变得更具美感，满足整体一致性的要求。主题的控制包括标题、坐标轴标签、图例标签等文字调整，以及网格线、背景、轴须的颜色搭配。

`lattice` 和基础图形系统没有采用数据与非数据控制分离的方法，大部分函数都设定了许多参数来调整数据和非数据的外观，这很容易导致函数的复杂化，使得图形的学习变得更为困难。`ggplot2` 则采用了不同的策略：绘图时，首先确定数据如何展示，然后再用主题系统对细节进行渲染。图 8.1 展示了改变主题后的部分效果，它们是 `ggplot2` 中两种默认的主题。

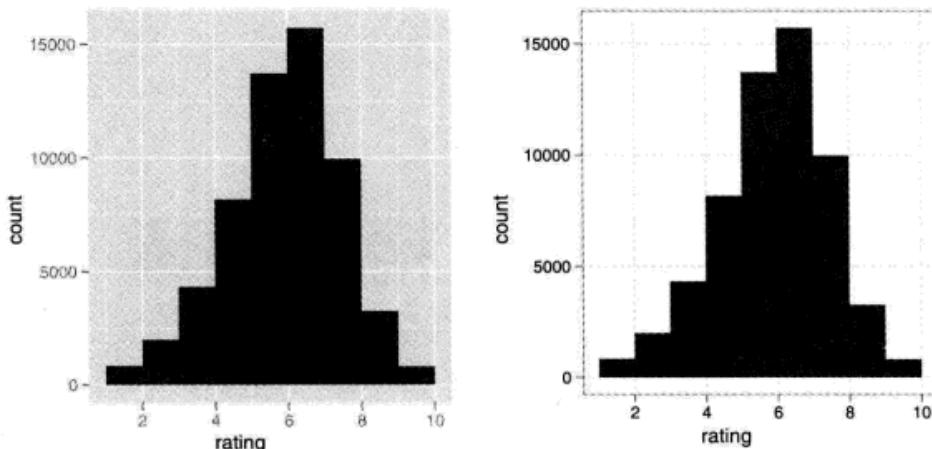


图 8.1 主题改变的效果。(左) 默认的灰色主题：灰色背景，白色网格线。(右) 黑白主题：白色背景，灰色网格线。两幅图中条状图和数据部分是相同的。

与 ggplot2 其他部分类似，主题也可以通过对许多参数的控制使图形由粗糙变得美观：

- 使用内置主题，图形每个元素的效果保持着视觉的一致性，详见第 8.1.1 节。默认主题为灰色背景、白色网格线，另一个为白色背景、灰色网格线。
- 修改内置主题的某些元素，详见 8.2 节。每个主题都是由许多元素组成，它们都沿袭了内置主题中渲染元素的函数和参数。通过调整和组合这些参数，比如文本的大小和颜色、背景、网格线颜色、文本方向等，你便可以获得自己独有的主题。

一般来说，每种主题设置都可进行全局性的修改，使之应用到所有图形中，或者进行局部性的修改，使之应用到某幅图形中。余下章节将对这些操作进行介绍。

8.1.1 内置主题

内置主题有两种。默认的 `theme_gray()` 使用淡灰色背景和白色网格线，建议来自于 Tufte(1990,1997,2001,2006) 和 Brewer(1994a); Carr(1994,2002); Carr 和 Sun(1999)。Cleveland(1993b) 认为网格线有助于图形位置的微调，但由于对视觉的冲击效果小，我们也可以删除它。从印刷角度来说，灰色背景的图形与文本有相似的色调，显得比较和谐，不会有文本从白色背景中蹦出来的突兀感，因此这种色调的连续性会使图形看起来具有整体感。另一个固定主

题 `theme_bw()` 为传统的白色背景和深灰色的网格线。图 8.1 展示了两个主题间的异同。

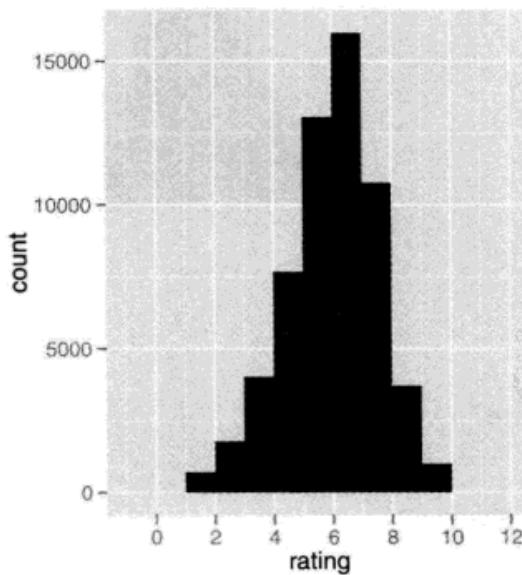
两个主题都由唯一的参数 `base_size` 来控制基础字体的大小。基础字体大小指的是轴标题的大小，图形标题比它大 20%，轴须标签比它小 20%。若你想对这些字体大小分别进行控制，下面的章节将会告诉你如何对它们进行修改。

主题设置有两种方式：

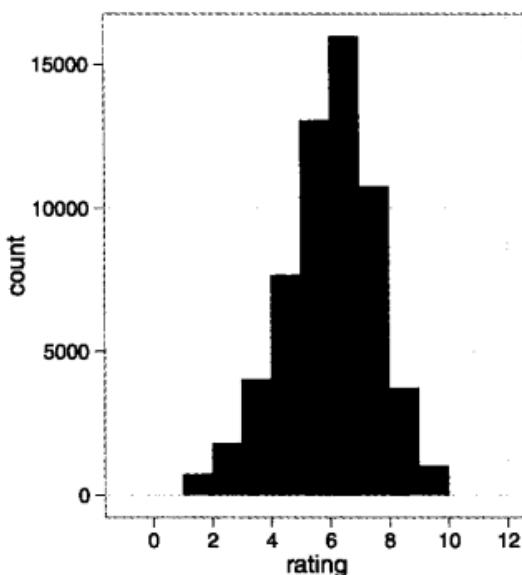
- 全局性设置：`theme_set(theme_grey())` 或 `theme_set(theme_bw())`。
`theme_set()` 返回先前的主题，可储存以备后用。
- 局部性设置：只改变单个图形的主题，`qplot(...)+ theme_grey()`。局部设置将会覆盖默认的全局性设置。

下面的代码给出了一些主题组合应用的例子：

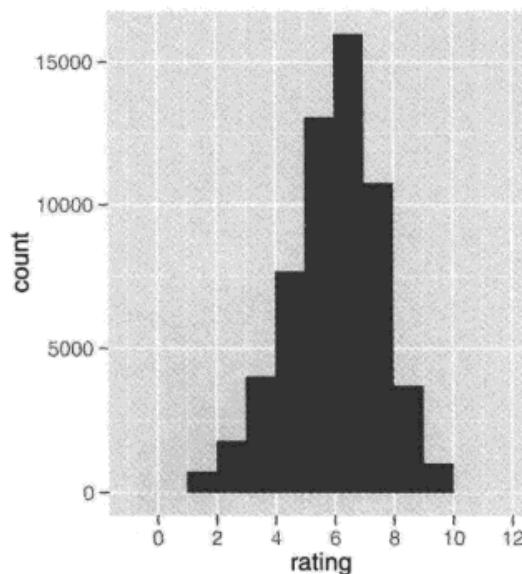
```
1 hgram <- qplot(rating, data = movies, binwidth = 1)
2 ## 主题在被创建时对图形无影响
3 ## 在被绘制时才会影响图形
4 hgram
```



```
1 previous_theme <- theme_set(theme_bw())
2 hgram
```



```
1 ## 你可以通过向图形添加主题来覆盖单个图形中的初始主题。  
2 ## 此处我们使用的是初始主题。  
3 hgram + previous_theme
```



```
1 ## 永久性储存初始主题  
2 theme_set(previous_theme)
```

8.1.2 主题元素和元素函数

主题由控制图形外观的多个元素组成，见表 8.1。有三个元素含有 `x` 和 `y` 的设置：`axis.text`、`axis.title` 和 `strip.text`。通过对水平和竖直方向元素的不同设置，我们可控制不同方向的文本外观，这些控制元素外观的函数被称为元素函数。

表 8.1 主题元素。

主题元素	类型	描述
axis.line	segment	直线和坐标轴
axis.text.x	text	x 轴标签
axis.text.y	text	y 轴标签
axis.ticks	segment	轴须标签
axis.title.x	text	水平轴标题
axis.title.y	text	竖直轴标题
legend.background	rect	图例背景
legend.key	rect	图例符号
legend.text	text	图例标签
legend.title	text	图例标题
panel.background	rect	面板背景
panel.border	rect	面板边界
panel.grid.major	line	主网格线
panel.grid.minor	line	次网格线
plot.background	rect	整个图形背景
plot.title	text	图形标题
strip.background	rect	分面标签背景
strip.text.x	text	水平条状文本
strip.text.y	text	竖直条状文本

内置元素函数有四个基础类型：文本 (text)、线条 (lines)、矩形 (rectangles)、空白 (blank)。每个元素函数都有一系列控制外观的参数。

- `element_text()` 绘制标签和标题，可控制字体的 `family`, `face`, `colour`, `size`, `hjust`, `vjust`, `angle`, `lineheight`。下面的代码和图 8.2 展示了不同参数下图形的变化情况。角度的改变可能对轴须标签很有用，当改变角度时，需将 `hjust` 调整至 0 或 1。

```

1 hgramt <- hgram + labs(title = "This is a histogram")
2 hgramt
3 hgramt + theme(plot.title = element_text(size = 20))

```

```

4 hgramt + theme(plot.title = element_text(size = 20, colour = "red"))
5 hgramt + theme(plot.title = element_text(size = 20, hjust = 0))
6 hgramt + theme(plot.title = element_text(size = 20, face = "bold"))
7 hgramt + theme(plot.title = element_text(size = 20, angle = 180))

```

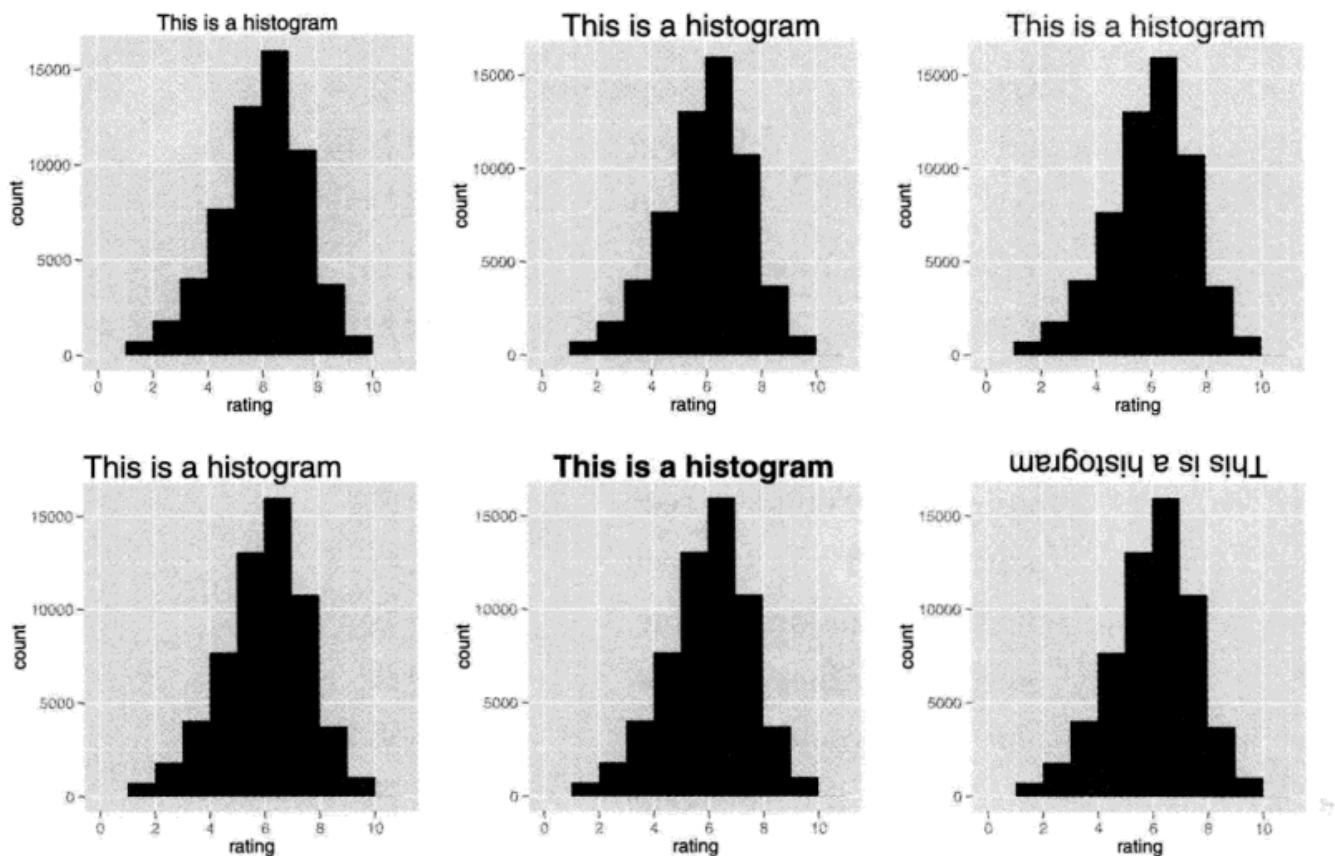


图 8.2 改变图形标题的外观。

- `element_line()` 绘制线条或线段，该元素函数可控制 `colour`, `size`, `linetype`。图 8.3 展示了相应结果，代码如下所示。

```

1 hgram + theme(panel.grid.major = element_line(colour = "red"))
2 hgram + theme(panel.grid.major = element_line(size = 2))
3 hgram + theme(panel.grid.major = element_line(linetype = "dotted"))
4 hgram + theme(axis.line = element_line())
5 hgram + theme(axis.line = element_line(colour = "red"))
6 hgram + theme(axis.line = element_line(size = 0.5,
7   linetype = "dashed"))

```

- `element_rect()` 绘制主要供背景使用的矩形。你可以控制填充颜色 (`fill`) 和边界的 `colour`, `size`, `linetype`，实例如图 8.4 所示，代码如下。

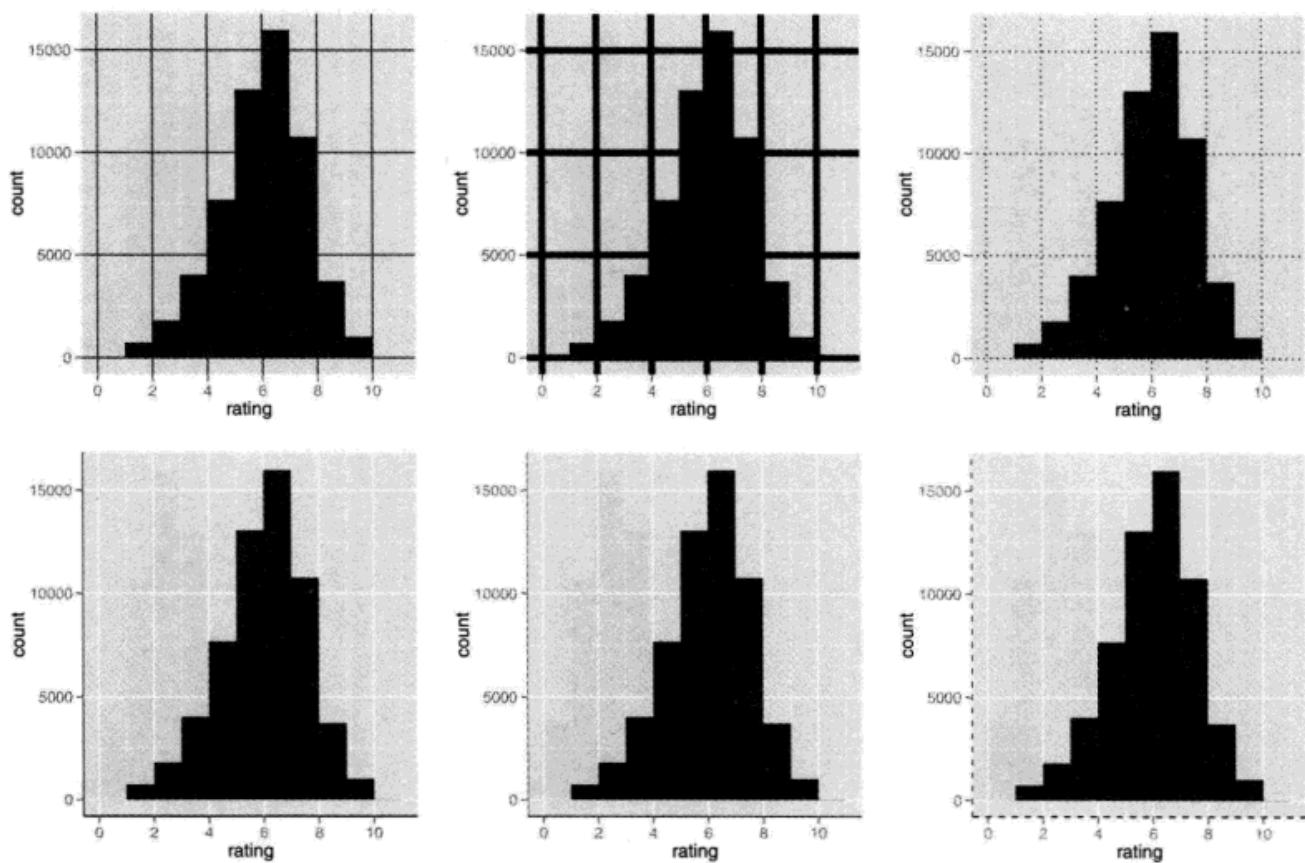


图 8.3 改变图形中线条和线段的外观。

```

1 hgram + theme(plot.background = element_rect(fill = "grey80", colour = NA))
2 hgram + theme(plot.background = element_rect(size = 2))
3 hgram + theme(plot.background = element_rect(colour = "red"))
4 hgram + theme(panel.background = element_rect())
5 hgram + theme(panel.background = element_rect(colour = NA))
6 hgram + theme(panel.background = element_rect(linetype = "dotted"))

```

- `element_blank()` 表示空主题，即对元素不分配相应的绘图空间。该函数可删去我们不感兴趣的绘图元素，如图 8.5。使用之前的 `colour = NA`, `fill = NA` 让某些元素不可见，可达到相同的效果，但仍占绘图空间。

```

1 hgramt
2 last_plot() + theme(panel.grid.minor = element_blank())
3 last_plot() + theme(panel.grid.major = element_blank())
4 last_plot() + theme(panel.background = element_blank())
5 last_plot() + theme(axis.title.x = element_blank(),

```

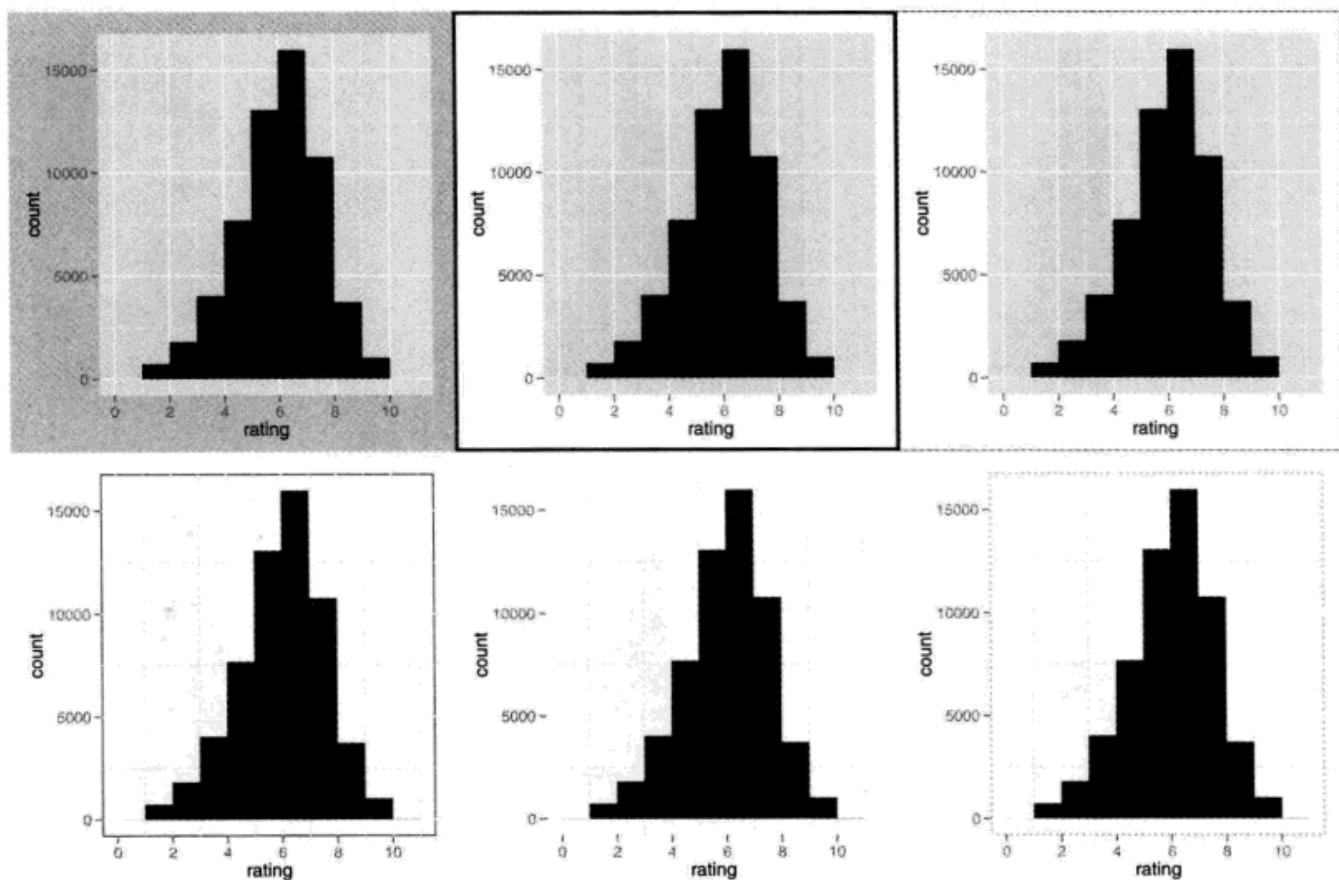


图 8.4 改变图形和面板背景的外观。

```

6   axis.title.y = element_blank()
7   last_plot() + theme(axis.line = element_line())

```

使用 `theme_get()` 可得到当前主题的设置。`theme()` 可在一幅图中对某些元素进行局部性地修改，`theme_update()` 可为后面图形的绘制进行全局性地修改。图 8.6 展示了多个主题设置结合起来的效果。

```

1 old_theme <- theme_update(
2   plot.background = element_rect(fill = "#3366FF"),
3   panel.background = element_rect(fill = "#003DF5"),
4   axis.text.x = element_text(colour = "#CCFF33"),
5   axis.text.y = element_text(colour = "#CCFF33", hjust = 1),
6   axis.title.x = element_text(colour = "#CCFF33", face = "bold"),
7   axis.title.y = element_text(colour = "#CCFF33", face = "bold",
8     angle = 90)
9 )
10 qplot(cut, data = diamonds, geom = "bar")

```

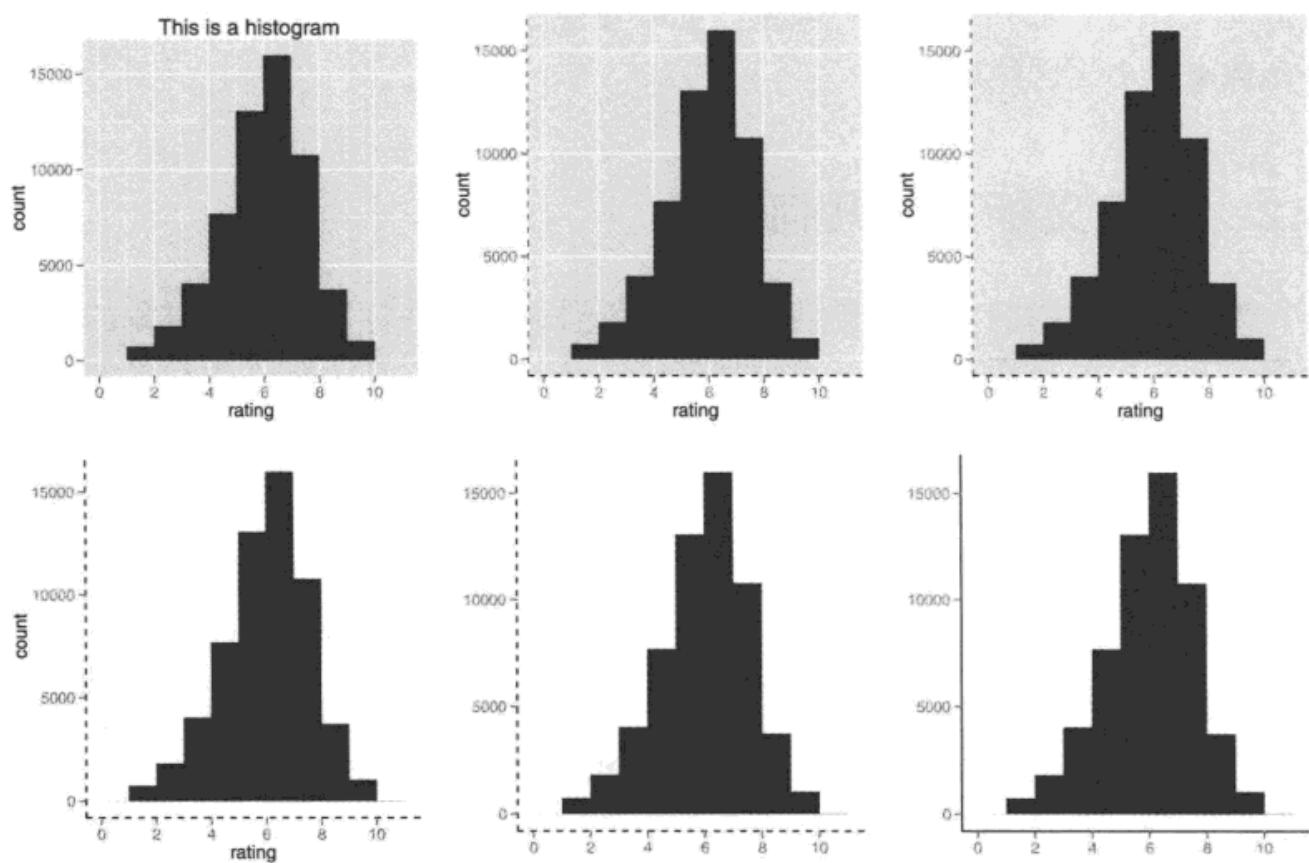


图 8.5 使用 element_blank() 逐步从图中删除非数据元素。

```
11 qplot(cty, hwy, data = mpg)
12 theme_set(old_theme)
```

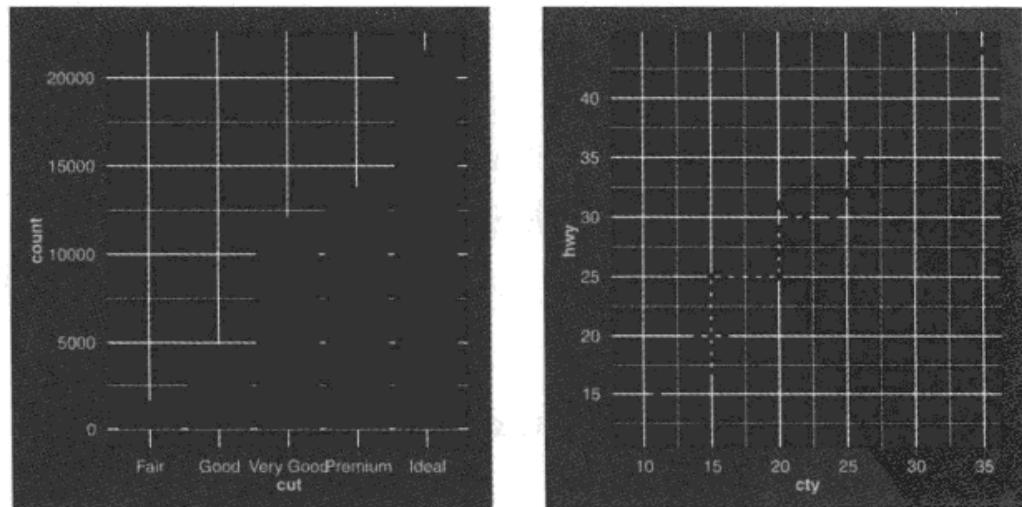


图 8.6 (参见彩图 32) 新主题中的条状图和散点图。

由于必须分别设置 x 和 y 元素，示例中的设置显得有些重复。但你想完全

控制元素外观，必须承受这种繁琐。如果你想设定自己独有的主题，最好写一个函数来最小化这种重复。

8.2 自定义标度和几何对象

为保证新主题的整体连贯性，你可能需要调整一些标度和几何对象的默认设置。下文中的函数可避免每次添加标度或几何对象时都不得不手动设置参数的繁琐。

8.2.1 标度^①

如果需要改变默认标度的显示方法，请重新定义或者用 `ggplot2` 中已经定义的可选方法覆盖控制相应图形属性的标度函数。该函数的命名方法通常是 `scale_aesthetics_continuous` 或者 `scale_aesthetics_discrete`，请将函数命名中的 `aesthetics` 替换成相应的图形属性，比如 `color`, `fill`, `size` 等等。示例如下：

```
1 p <- qplot(mpg, wt, data = mtcars, colour = factor(cyl))
2 p
3
4 scale_colour_discrete <- scale_colour_brewer
5 p
```

8.2.2 几何对象和统计变换

类似 `update_geom_defaults()` 和 `update_stat_defaults()`，我们也可以自定义几何对象和统计变换。与其他的主题设置不同，此处设置只影响设置改变后新绘制的图形，而不是所有图形。下面例子展示了如何改变默认颜色和将默认直方图改造成密度直方图。

```
1 update_geom_defaults("point", aes(colour = "darkblue"))
2 qplot(mpg, wt, data=mtcars)
3 update_stat_defaults("bin", aes(y = ..density..))
4 qplot(rating, data = movies, geom = "histogram", binwidth = 1)
```

^①本节为适应 0.9.0 及之后版本而重写

表 8.2 列出了所有常见的图形属性默认设置。如果你改变了一个几何对象的默认值，那么最好也要改变其他所有几何对象的默认值，以保持图形的整体一致性。附录 B 列出了所有颜色、曲线类型、形状和大小设置的细节，可参考它来调整参数。

表 8.2 几何对象的图形属性的默认设置，更多细节可参见附录 B。

图形属性	默认值	几何对象
colour	#3366FF	contour, density2d, quantile, smooth
colour	NA	area, bar, histogram, polygon, rect, tile
colour	black	abline, crossbar, density, errorbar, hline, line, linerange, path, pointrange, rug, segment, step, text, vline
colour	darkblue	jitter, point
colour	grey60	boxplot, ribbon
fill	NA	crossbar, density, jitter, point, pointrange
fill	grey20	area, bar, histogram, polygon, rect, ribbon, tile
linetype	1	abline, area, bar, contour, crossbar, density, density2d, errorbar, histogram, hline, line, linerange, path, pointrange, polygon, quantile, rect, ribbon, rug, segment, smooth, step, tile, vline
shape	19	jitter, point, pointrange
size	0.5	abline, area, bar, boxplot, contour, crossbar, density, density2d, errorbar, histogram, hline, line, linerange, path, pointrange, polygon, quantile, rect, ribbon, rug, segment, smooth, step, vline
size	2	jitter, point
weight	1	bar, boxplot, contour, density, density2d, histogram, quantile, smooth

8.3 存储输出

基本图形输出有两种类型：矢量型或光栅型。矢量图是过程化的，意味着图形可无限缩放而没有细节的损失。光栅图以像素阵列形式存储，具有固定的最优观测大小。图 8.7 通过一个简单的圆圈阐释了两者的不同，更多细节描述可参考 <http://tinyurl.com/rstrvctr>。

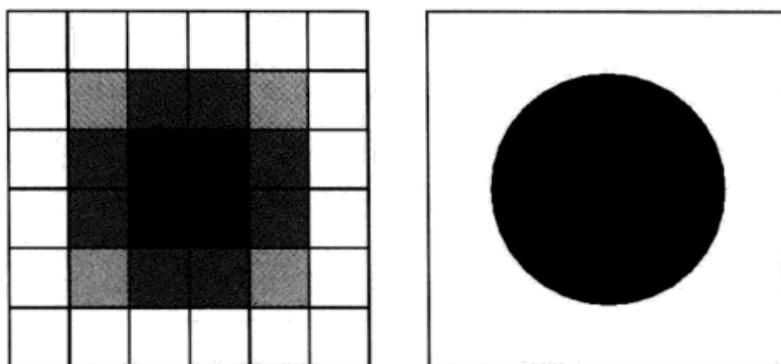


图 8.7 图解光栅图(左)和矢量图(右)的不同。

一般来说，矢量输出更受人喜爱，但对于包含数千个对象的图形，矢量渲染会变得很慢，此时最好转化为光栅输出。对于图形印刷，高分辨率（例如 600dpi）可能是较好的折中方案，但是图形可能会很大。

你可使用 R 基础的图形设备或者 ggplot2 中特有的函数 `ggsave()` 来存储当前图形。`ggsave()` 是为了图形交互而优化过的函数，它有如下重要的参数：

- `path` 设定图形存储的路径，它会根据文件扩展名自动选择正确的图形设备。
- 三个控制输出尺寸的参数。若空白，则使用当前屏幕图形设备尺寸。`width` 和 `height` 设置绝对尺寸大小，`scale` 设置图形相对屏幕展示的尺寸大小。当最终图形绘制好后，最好设置 `width` 和 `height`，这样可精确控制输出尺寸。
- 对于光栅图形，`dpi` 参数控制图形的分辨率，默认值为 300，适合大部分打印设备，但是你可以修改为 600，用于高分辨率输出，或者修改为 72，用于屏幕（如网页）展示。

下面的代码展示了两种存储方法。若你想将两幅图形存储到一个文件中，你需要打开基于磁盘的图形设备（比如 `png()` 或 `pdf()`），打印图形，然后关闭图形设备 `dev.off()`。

```
1 qplot(mpg, wt, data = mtcars)
2 ggsave(file = "output.pdf")
3 pdf(file = "output.pdf", width = 6, height = 6)
4 ## 在脚本中，你需要明确使用 print() 来打印图形
5 qplot(mpg, wt, data = mtcars)
```

```
6 qplot(wt, mpg, data = mtcars)
7 dev.off()
```

表 8.3 列出了推荐的图形格式。作为 Linux 开发工具链的一部分，在 L^AT_EX 文档中，R 使用 png 或者 pdf 输出表现最好。Microsoft Office 可以很好地支持高分辨率的 png 文件，但对于矢量输出，Windows 的图元文件和 postscript 都不支持半透明效果。虽然 postscript 打印效果好，但只有在其他的软件包中添加了预览功能才能将图形展现在屏幕上。半透明效果可用来展现置信区间，点将会穿过这个半透明区域。若将图形复制粘贴到 Word 或 OpenOffice 中，置信区间带就会消失。

若用 L^AT_EX，可在导言区添加 \DeclareGraphicsExtensions{.png, .pdf}，这样就不必用 \includegraphics 命令来设置图形扩展名，不过 L^AT_EX 将优先使用 png 而不是 pdf。之所以选择推荐这个顺序，是因为你可以先生成 pdf 文件，然后将太大的 pdf 文件重新渲染成 png 格式的文件。另外一个有用的命令是 \graphicspath{{include/}}，可用来设定寻找图形的路径，这样就可将图形和文本分目录管理。

表 8.3 不同功用的推荐图形设备。

软件	推荐的图形设备
Illustrator	svg
latex	ps
MS Office	png (600 dpi)
Open Office	png (600 dpi)
pdflatex	pdf, png (600 dpi)
web	png (72 dpi)

8.4 一页多图

若想绘制一页多图，你必须了解些 grid (ggplot2 中默认调用) 的工作原理。其关键概念是视图窗口 (viewport)：显示设备的一个矩形子区域。默认的视图窗口占据了整个绘图区域，通过设置视图窗口，你可以任意安排多幅图形的位置。

首先，我们绘制三幅测试图形。若想一页多图，最简单的方式是创建图形并将图形赋成变量，然后再绘制出来。这样你就只需关注图形的摆放而不是内容了。图 8.8 展示了下面的代码生成的图形。

```
1 (a <- qplot(date, unemploy, data = economics, geom = "line"))
2 (b <- qplot(uempmed, unemploy, data = economics) + geom_smooth(se = F))
3 (c <- qplot(uempmed, unemploy, data = economics, geom="path"))
```

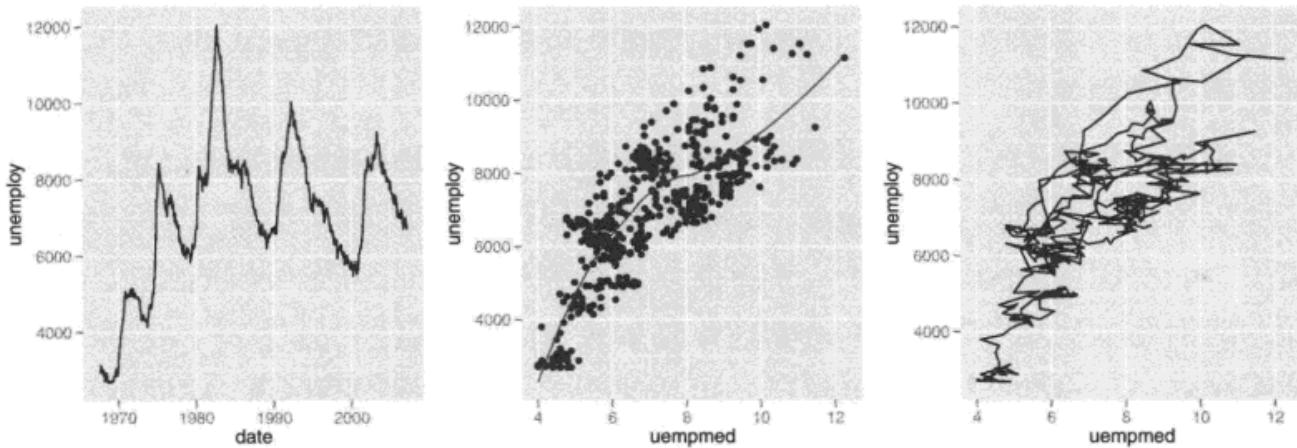


图 8.8 测试图形布局的三幅图形。

8.4.1 子图

把子图嵌入到主图的顶部是一种常见的图形布局。为达到这个效果，我们首先需要绘制主图，然后在更小的视图窗口绘制子图。`viewport()` 函数可创建视图窗口，参数 `x`, `y`, `width`, `height` 控制视图窗口的大小和位置。默认的测量单位是“`npc`”，范围从 0 到 1。`(0,0)` 代表的位置是左下角，`(1,1)` 代表右上角，`(0.5,0.5)` 代表视图窗口的中心。如果这些单位不符合你的需求，你也可使用如 `unit(2, "cm")` 或 `unit(1, "inch")` 这样的绝对单位。

```
1 library(grid)
2 ## 一个占据整个图形设备的视图窗口
3 vp1 <- viewport(width = 1, height = 1, x = 0.5, y = 0.5)
4 vp1 <- viewport()
5 ## 只占了图形设备一半的宽和高的视图窗口,
6 ## 定位在图形的中间位置
7 vp2 <- viewport(width = 0.5, height = 0.5, x = 0.5, y = 0.5)
8 vp2 <- viewport(width = 0.5, height = 0.5)
```

```

9 ## 一个 2cm x 3cm 的视图窗口, 定位在图形设备中心
10 vp3 <- viewport(width = unit(2, "cm"), height = unit(3, "cm"))

```

默认地, `x` 和 `y` 参数控制着视图窗口的中心位置, 若想调整图形位置, 你需要通过 `just` 参数来控制将图形放置在哪个边角。

```

1 ## 在右上角的视图窗口
2 vp4 <- viewport(x = 1, y = 1, just = c("top", "right"))
3 ## 处在左下角
4 vp5 <- viewport(x = 0, y = 0, just = c("bottom", "right"))

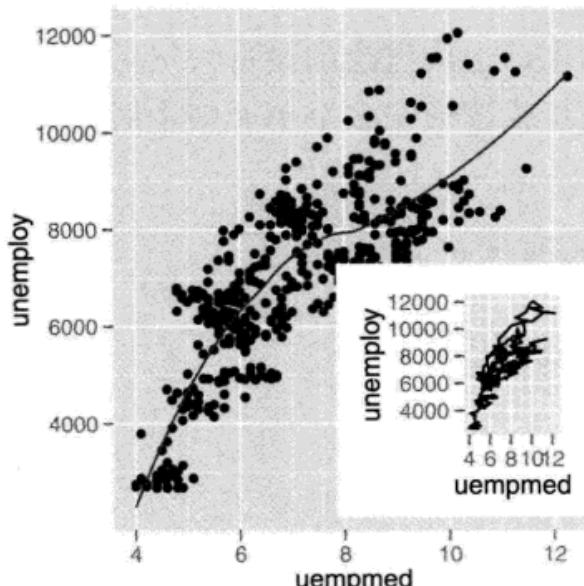
```

为了能在新的视图窗口中画图, 我们还需要使用 `print()` 中的 `vp` 参数。通常在命令行上解析某些代码时会自动调用 `print()`, 但因为我们想自定义视图窗口, 因此需要我们手动调用。图 8.9(a) 展示了以下代码的结果。

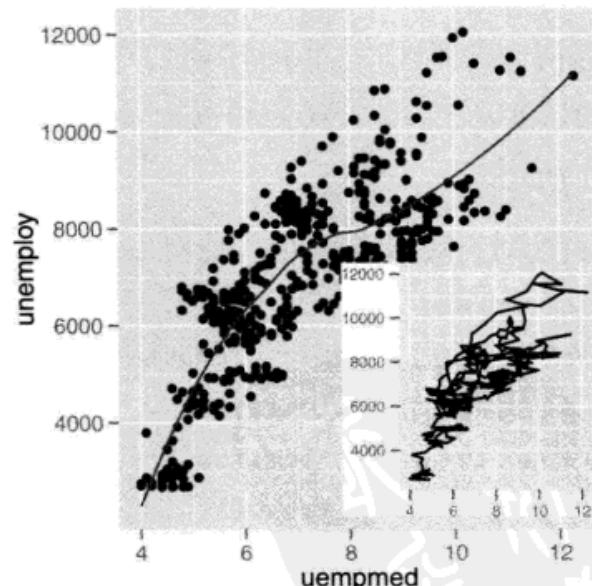
```

1 pdf("polishing-subplot-1.pdf", width = 4, height = 4)
2 subvp <- viewport(width = 0.4, height = 0.4, x = 0.75, y = 0.35)
3 b
4 print(c, vp = subvp)
5 dev.off()

```



(a) 带有子图的图形



(b) 为更好展示而微调子图

图 8.9 两个包含子图的图形实例。为得到最优的图形展示, 你通常需要微调主题设置。

这是我们预期的图形, 不过仍需对图形的外观进行一些微调: 文本应更小些, 要移除轴标签, 缩减图形边界。结果如图 8.9(b), 代码如下。

```
1 csmall <- c +
2   theme_gray(9) +
3   labs(x = NULL, y = NULL) +
4   theme(plot.margin = unit(rep(0, 4), "lines"))
5 pdf("polishing-subplot-2.pdf", width = 4, height = 4)
6 b
7 print(csmall, vp = subvp)
8 dev.off()
```

注意需要使用 `pdf()` 或 `png()` 将图形存储到磁盘中，因为 `ggsave()` 只能存储一幅图。

8.4.2 矩形网格

更复杂的情形是在矩形网格绘制大量图形。当然，你可以使用以上你学到的视窗图方法来解决，但大量的手工计算非常繁琐。一个更好的处理方式为 `grid.layout()`，它设置了一个任意高和宽的视图窗口网格。虽然你仍需一个个创建视图窗口，但不用设置视图窗口的位置和大小，只需设置布局 (`layout`) 的行数和列数即可。

下面的例子阐释了网格法的工作原理。首先创建一个布局，如 2×2 网格，随后将它分配到一个视图窗口中，在视图窗口上构建绘图设备，然后在网格预期的位置上绘图。我们可按行向量或列向量将图形在多个单元格中展开，如图 8.10 所示。

```
1 pdf("polishing-layout.pdf", width = 8, height = 6)
2 grid.newpage()
3 pushViewport(viewport(layout = grid.layout(2, 2)))
4 vplayout <- function(x, y)
5   viewport(layout.pos.row = x, layout.pos.col = y)
6 print(a, vp = vplayout(1, 1:2))
7 print(b, vp = vplayout(2, 1))
8 print(c, vp = vplayout(2, 2))
9 dev.off()
```

在默认的 `grid.layout()` 中，每个单元格的大小都相同，你可设置 `widths` 和 `heights` 参数使它们具有不同的大小。具体例子请参考 `grid.layout()` 的文档。

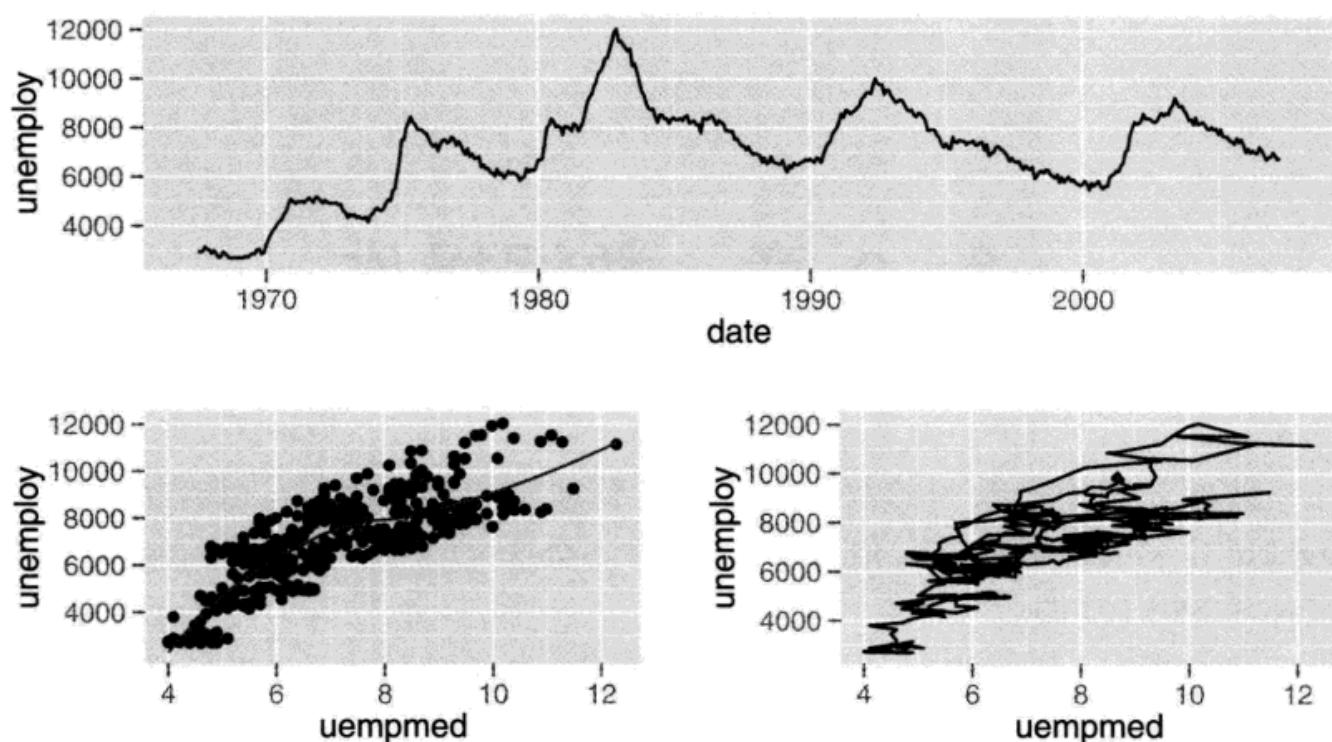


图 8.10 `grid.layout()` 将三幅图形分置在一页上。

第 9 章 数据操作

前面几章中涉及到的数据都是已经整理好的数据框，可以直接输入到 `ggplot()` 或 `qplot()` 中。但是实际数据并不会这么理想，通常需要经过整理和变换才能用于作图。

本章 9.1 节会谈到如何用 `plyr` 包实现 `ggplot2` 内部的统计变换过程。9.2 节会提到时间序列图和平行坐标图里很有用的长数据。9.3 节介绍如何用 R 对象作图（而不是数据框），以及如何用 `ggplot2` 实现灵活的线性模型诊断。

事实上，数据清洗、操作和变换是一个很大的问题，本章只涉及其中跟 `ggplot2` 有关的一点皮毛。以下参考文献对此进行了更深入的探讨：

- *Data Manipulation with R* (Phil Spector, Springer, 2008);
- *plyr: divide and conquer for data analysis*, Hadley Wickham. 可在 <http://had.co.nz/plyr> 免费下载。这个文档是对 `plyr` 包的一个完整的描述。
- *Reshaping data with the reshape package*, Hadley Wickham. *Journal of Statistical Software*, 21(12), 2007. <http://www.jstatsoft.org/v21/i12/>. 这是对 9.2 节中 `melt` 函数更完整的一个介绍。这个函数可以对数据进行一系列的汇总和变形，功能类似于数据透视表。

9.1 `plyr` 包简介

使用分面 (`faceting`) 参数，`ggplot2` 可以同时在数据的多个子集上作出相同的图。相应地，`plyr` 包中的 `ddply()` 函数能够同时在数据的多个子集上作统计汇总。`plyr` 包提供了一整套工具集来处理列表 (`list`)、数组 (`array`) 和数据框 (`data.frame`)，把复杂的数据分割成几个部分，分别对各个部分进行处理，最后把这些结果综合到一起。这里主要用到其中的 `ddply()` 函数。该函数能够

根据行的取值，把数据框分解成几个子集，分别把各个子集输入某个函数，最后把结果综合在一个数据框内。函数基本用法是 `ddply(.data, .variables, .fun, ...)`，其中：

- `.data` 是用来作图的数据。
- `.variables` 是对数据取子集的分组变量，形式是 `.(var1, var2)`。为了与图形保持一致，该变量必须包含所有在画图过程中用到的分组变量和分面变量 (faceting variable，分面变量用来把数据分割成几个部分，每个部分分别画在一张小图里；而分组变量则是同一张图内的数据分成几个部分处理)。
- `.fun` 是要在各子集上运行的统计汇总函数。这个函数可以返回向量也可以返回数据框。注意这里不要求 `.fun` 的返回结果包含分组变量，如果需要的话分组变量会被自动加入到最终结果里。函数返回结果可以是高度概括的数据集 (也许只是一个数)，也可以是用修改或扩展之后的数据。

更多的信息和例子参见函数的文档 (`?ddply`) 和 `plyr` 包的网站 <http://had.co.nz/plr>。以下例子演示了一些常见的数据操作及其需要使用的统计汇总函数。

- `subset()` 用来对数据取子集的函数，选择数据中前 (或后) n 个 (或 $x\%$) 的观测值，或是在某个阈值之上或下的观测值：

```
1 ## 选取各个颜色里最小的钻石
2 ddply(diamonds, .(color), subset, carat == min(carat))
3 ## 选取最小的两颗钻石
4 ddply(diamonds, .(color), subset, order(carat) <= 2)
5 ## 选取每组里大小为前 1% 的钻石
6 ddply(diamonds, .(color), subset, carat > quantile(carat, 0.99))
7 ## 选出所有比组平均值大的钻石
8 ddply(diamonds, .(color), subset, price > mean(price))
```

- `transform()` 是用来进行数据变换的函数，与 `ddply()` 一起可以计算分组统计量，例如各组的标准差，并且加到原数据上去。另外，9.2.1 小节还介绍了这个函数的另一种用法——把几个时间序列数据调整到同一标度。

```
1 ## 把每个颜色组里钻石的价格标准化，使其均值为 0，方差为 1
2 ddply(diamonds, .(color), transform, price = scale(price))
```

```

3 ## 减去组均值
4 ddply(diamonds, .(color), transform, price = price - mean(price))

```

- `colwise()` 用来向量化一个普通函数，也就是说 `colwise()` 能把原本只接受向量输入的函数变成可以接受数据框输入的函数。要注意 `colwise()` 返回的是一个新的函数而不是函数运行结果。下面例子里函数 `nmissing()` 计算向量里缺失值的数目，用 `colwise()` 向量化后，可以应用到数据框上，计算数据框各列的缺失值数目。

```

1 > nmissing <- function(x) sum(is.na(x))
2 > nmissing(msleep$name)
3 [1] 0
4 > nmissing(msleep$brainwt)
5 [1] 27
6 >
7 > nmissing_df <- colwise(nmissing)
8 > nmissing_df(msleep)
9   name genus vore order conservation sleep_total sleep_rem
10  1     0     0     7     0         29          0        22
11   sleep_cycle awake brainwt bodywt
12  1       51     0     27      0
13 > ## 更便捷的方法
14 > colwise(nmissing)(msleep)
15   name genus vore order conservation sleep_total sleep_rem
16  1     0     0     7     0         29          0        22
17   sleep_cycle awake brainwt bodywt
18  1       51     0     27      0

```

`numcolwise()` 是 `colwise()` 的一个特殊版本，功能类似，但 `numcolwise()` 只对数值类型的列操作。例如，`numcolwise(median)` 对每个数值类型的列计算中位数，`numcolwise(quantile)` 对每个数值类型的列计算分位数。有了 `numcolwise()`，相应地就有 `catcolwise()`，`catcolwise()` 只对分类类型的列操作。

```

1 > msleep2 <- msleep[, -6]    ## 移除第六列
2 > numcolwise(median)(msleep2, na.rm = T)
3   sleep_rem sleep_cycle awake brainwt bodywt
4   1       1.5  0.3333333  13.9  0.0124   1.67

```

```

5 > numcolwise(quantile)(msleep2, na.rm = T)
6   sleep_rem sleep_cycle awake brainwt bodywt
7 1      0.1  0.1166667  4.10 0.00014    0.005
8 2      0.9  0.1833333 10.25 0.00290    0.174
9 3      1.5  0.3333333 13.90 0.01240    1.670
10 4     2.4  0.5791667 16.15 0.12550   41.750
11 5     6.6  1.5000000 22.10 5.71200 6654.000
12 > numcolwise(quantile)(msleep2, probs = c(0.25, 0.75),
13 + na.rm = T)
14   sleep_rem sleep_cycle awake brainwt bodywt
15 1      0.9  0.1833333 10.25 0.0029  0.174
16 2     2.4  0.5791667 16.15 0.1255 41.750

```

以上这些函数与 ddply 一起可以对数据进行各种分组统计。

```

1 > ddply(msleep2, .(vore), numcolwise(median), na.rm = T)
2   vore sleep_rem sleep_cycle awake brainwt bodywt
3 1 carni      1.95  0.3833333 13.6 0.044500 20.490
4 2 herbi      0.95  0.2166667 13.7 0.012285  1.225
5 3 insecti     3.00  0.1666667  5.9 0.001200  0.075
6 4 omni       1.85  0.5000000 14.1 0.006600  0.950
7 5 <NA>       2.00  0.1833333 13.4 0.003000  0.122
8 > ddply(msleep2, .(vore), numcolwise(mean), na.rm = T)
9   vore sleep_rem sleep_cycle   awake   brainwt   bodywt
10 1 carni  2.290000  0.3733333 13.62632 0.07925556 90.75111
11 2 herbi  1.366667  0.4180556 14.49062 0.62159750 366.87725
12 3 insecti 3.525000  0.1611111  9.06000 0.02155000 12.92160
13 4 omni   1.955556  0.5924242 13.07500 0.14573118 12.71800
14 5 <NA>  1.880000  0.1833333 13.81429 0.00762600  0.85800

```

- 如果以上提供的函数不够用，读者可以编写自己的函数，只要它也能接收、输出数据框就可以。例如下面的函数用于计算价格和克拉的秩相关系数以及对数变换后的普通相关系数。

```

1 > my_summary <- function(df) {
2 +   with(df, data.frame(
3 +     pc_cor = cor(price, carat, method = "spearman"),
4 +     lpc_cor = cor(log(price), log(carat)))
5 +   ))

```

```
6 + }
7 > ddply(diamonds, .(cut), my_summary)
8
9     cut    pc_cor    lpc_cor
10 1     Fair  0.9056144 0.9085131
11 2     Good  0.9599684 0.9687510
12 3 Very Good  0.9688534 0.9716746
13 4   Premium  0.9605332 0.9697578
14 5     Ideal  0.9537275 0.9661884
15
16 > ddply(diamonds, .(color), my_summary)
17   color    pc_cor    lpc_cor
18 1      D  0.9561208 0.9606617
19 2      E  0.9600994 0.9643845
20 3      F  0.9641572 0.9623876
21 4      G  0.9633681 0.9696785
22 5      H  0.9730390 0.9801569
23 6      I  0.9834392 0.9865118
24 7      J  0.9846710 0.9879449
```

注意，上面例子里的函数 `my_summary()` 里没有输出分组变量 `cut` 和 `color`，但是最终输出里却自动包含了这些分组变量。

以上所有问题都有个共同点，就是这些操作对单独的子集很简单，也许只需要一行代码。但针对多个子集操作并且组合各个结果就相对来说麻烦些。有了 `ddply()`，这些操作就可以直接完成了。

下面的这个案例介绍怎样用 `plyr` 完成 `ggplot2` 内置的统计功能。在某些情况下，例如需要把统计结果存到硬盘上或是想把这个方法用到其他数据集上的时候，这么做是很有必要的。而且这样还能够检查 `ggplot2` 的中间过程与我们的想法是否完全一致。

9.1.1 拟合多个模型

本节演示 `stat_smooth` 生成平滑数据 (smoothed data) 的过程 (“平滑”是指用一个函数来拟合数据间的关系，例如回归就是一种平滑过程。而“平滑数据”是指用平滑函数得到的预测值)。事实上，不仅仅是平滑数据，这个过程对

任何其他统计量都是一样的，而且允许在 ggplot2 之外定制更复杂的方法。图 9.1 是以下代码生成的分组平滑结果。

```

1 qplot(carat, price, data = diamonds, geom = "smooth",
2   colour = color)
3 dense <- subset(diamonds, carat < 2)
4 qplot(carat, price, data = dense, geom = "smooth",
5   colour = color, fullrange = TRUE)

```

那么如何用 plyr 生成一模一样的图呢？首先读下 `stat_smooth()` 的文档，看看它内部用的是什么模型：对于大数据它用的是 `gam(y ~ s(x, bs = "cs"))`。为了得到与 `stat_smooth` 同样的结果，先拟合这个模型，然后在一系列等距的点上求模型的预测值。这个过程是用下面代码中的 `smooth()` 函数实现的。有了这个函数，再结合 `ddply()` 就可以对不同颜色的钻石画出平滑曲线。

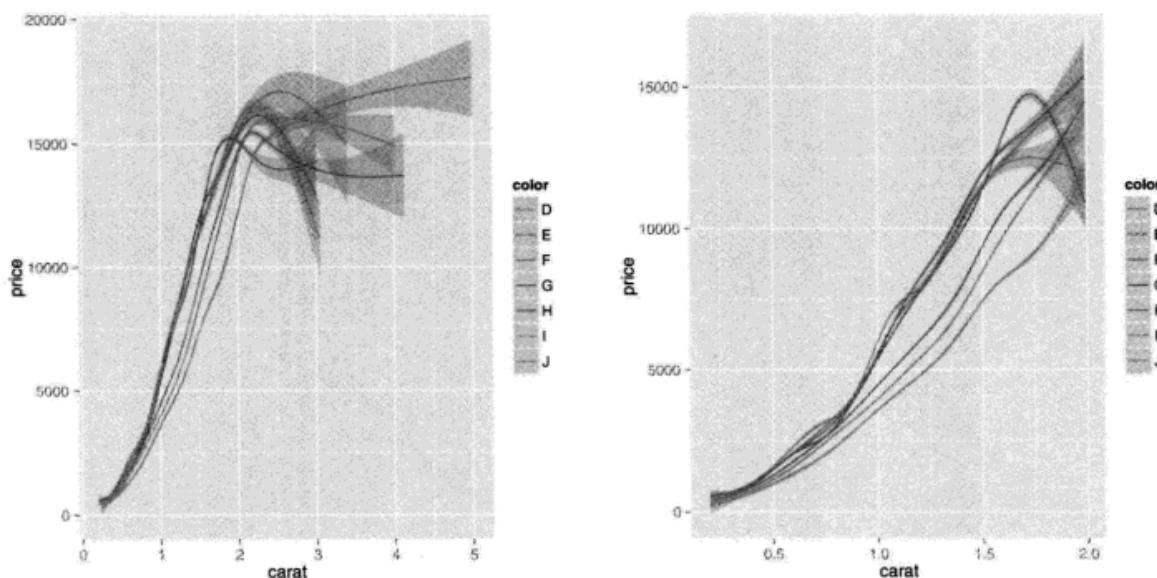


图 9.1 (参见彩图 33) 图中是不同颜色钻石 `price` 对 `carat` 的平滑趋势。左图是全部数据，可以看到重量大于 2 克拉后，标准差就激增，因为重量大于 2 克拉的钻石的个数较少。右图我们只画重量小于 2 克拉的钻石，重点关注数据较多的区域。

图 9.2 是结果，与 ggplot2 得到的一样。

```

1 library(mgcv)
2 smooth <- function(df) {
3   mod <- gam(price ~ s(carat, bs = "cs"), data = df)
4   grid <- data.frame(carat = seq(0.2, 2, length = 50))

```

```

5 pred <- predict(mod, grid, se = T)
6 grid$price <- pred$fit
7 grid$se <- pred$se.fit
8 grid
9 }
10 smoothes <- ddply(dense, .(color), smooth)
11 qplot(carat, price, data = smoothes, colour = color,
12   geom = "line")
13 qplot(carat, price, data = smoothes, colour = color,
14   geom = "smooth", ymax = price + 2 * se, ymin = price - 2 * se)

```

如果分组变量也被纳入了平滑模型的话，那么用 `plyr` 完成这个过程可以更灵活些。例如以下模型把钻石的价格当作克拉数和颜色的非线性函数。当然这个模型很不理想，因为它可能会把某些小且劣质的钻石价格预测为负。

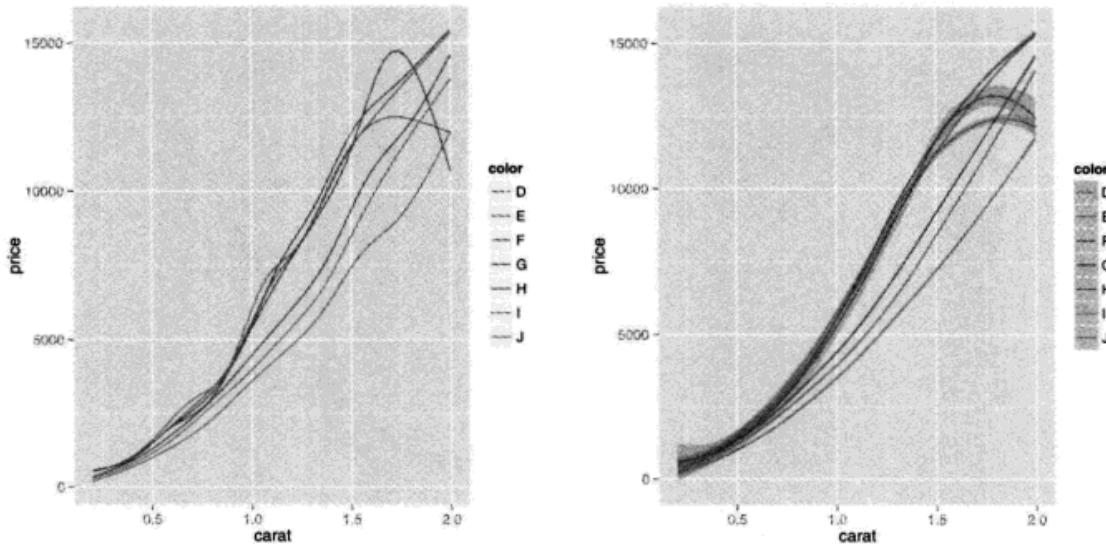
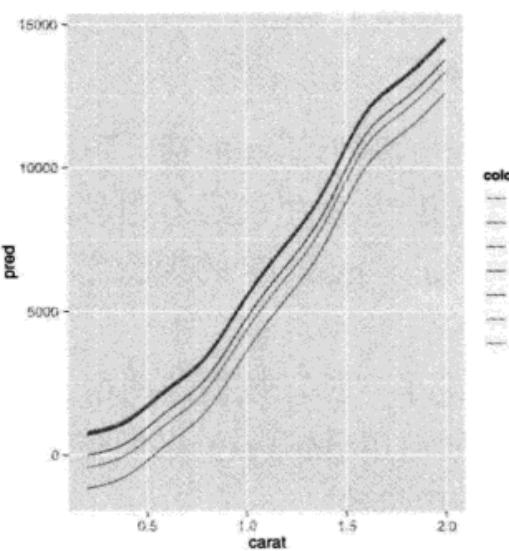


图 9.2 (参见彩图 34) 手动计算图 9.1 中的统计量。左图为预测值，右图加上标准差。

```

1 > mod <- gam(price ~ s(carat, bs = "cs") + color, data = dense)
2 > grid <- with(diamonds, expand.grid(
3 +   carat = seq(0.2, 2, length = 50),
4 +   color = levels(color)
5 + ))
6 > grid$pred <- predict(mod, grid)
7 > qplot(carat, pred, data = grid, colour = color, geom = "line")

```



组合多个模型的其他方法参见 4.9.3 节和 5.8 节。

9.2 把数据化“宽”为“长”

`ggplot2` 进行数据分组时必须根据行，而不能根据列。例如可以把钻石根据颜色分组，却不能把钻石的克拉数和价格变量分成两组。所以在一张图中以每种颜色的钻石为一组，分别画价格曲线很容易，但想把价格和克拉数两个变量作为两组，分别画一条曲线就很难。本节介绍如何把“宽”的数据变成“长”的数据，也就是说变量不再是放在各个列上，而是排成一列，每个变量都分别占其中的几行。这样就能够根据行把每个变量分组，然后作图。

本节将会使用到 `reshape2` 包^① (Wickham, 2007) 中的 `melt()` 函数。另外，`reshape2` 包还提供了 `cast()` 函数，它能灵活地整理、综合数据。感兴趣的读者可以进行深入地了解。表 9.1 给出了一个例子。`melt()` 函数有三个参数：

- `data`: 待变形的原数据。例如表 9.1 中左表的数据。
- `id.vars`: 依旧放在列上、位置保持不变的变量。`id.vars` 通常是离散的，并且是预先给定的。用方差分析的角度来理解，`id.vars` 好比是变量 Y_{ijk} 的下标 (i, j, k) ；用数据库的角度来理解，`id.vars` 好比是数据表的主键。例如表 9.1 中的 `date` 变量，变形后依旧单独占一列。
- `measure.vars`: 需要被放进同一列的变量。不同的变量放在同一列后，根据原变量名来分组，这样这些变量就可以同时画在一张图里。例如表 9.1

^①译者注：0.9.0 版本后使用 `reshape2`，而不是 `reshape`。

中变量 `pce` 和 `pop` 都被放在了 `value` 列里，用新增的 `variable` 用来记录每行数据原来属于哪一列。

如果读者熟悉 Wilkinson 的图形语法的话，可能想知道为什么其中的代数部分在 `ggplot2` 里没有。这是因为 R 里已经有很多的数据变形方法了，没有必要在 `ggplot2` 里增加这样的功能。`ggplot2` 的理念之一就是把数据变形和可视化尽可能分开。

下面的几节更详细地讨论了“长”数据的两个重要应用：多重时间序列图和平行坐标图。你将会看到如何用 `dplyr()` 来调整变量标度，并学习到在处理这类数据时 `ggplot2` 的几个有用特性。

9.2.1 多重时间序列

以下以 `economics` 数据为例。该数据包含诸如失业人数 (`unemploy`) 和失业周数的中位数 (`uempmed`) 等月度经济数据，我们预判这两个变量可能会相关。图 9.3 把两个变量画在两个单独的趋势图里，还有两者的散点图。但是如果想在一张图上同时画这两个时间序列的趋势图呢？

表 9.1 表中是 `economics` 数据“宽”的形式(左图)和“长”的形式(右图)。两个表中的数据是完全一样的，但是组织方法不同。`ggplot2` 用“长”数据对每个变量作图更方便。

			date	variable	value
			1967-06-30	pce	508
			1967-07-31	pce	511
			1967-08-31	pce	517
date	pce	pop	1967-09-30	pce	513
1967-06-30	508	198,712	1967-10-31	pce	518
1967-07-31	511	198,911	1967-11-30	pce	526
1967-08-31	517	199,113	1967-06-30	pop	198,712
1967-09-30	513	199,311	1967-07-31	pop	198,911
1967-10-31	518	199,498	1967-08-31	pop	199,113
1967-11-30	526	199,657	1967-09-30	pop	199,311
			1967-10-31	pop	199,498
			1967-11-30	pop	199,657

一种方法是用 6.4.4 节的方法，画图时把两个变量放在两个不同的图层上。

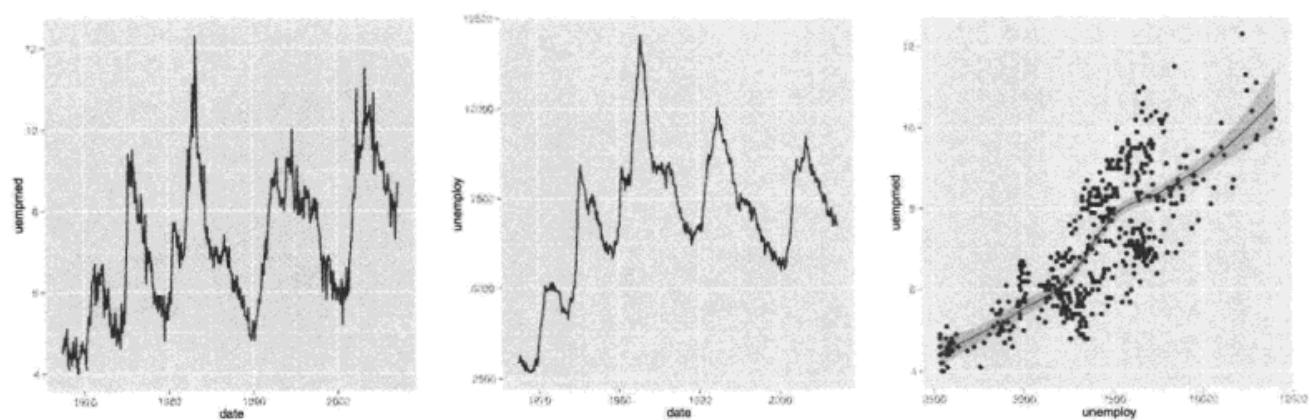


图 9.3 如果 `economics` 数据是“宽”的格式时，很容易分别画出两个变量的时间序列图（左图和中图），也很容易画出散点图（右图）。

但是如果有很多变量的话，这样做就很繁琐了。比较好的方法是，把数据变成一个“长”的格式，然后再可视化。在“长”数据里，时间序列值存在 `value` 变量里，然后用 `variable` 变量来区分。下面的代码就演示了这两种方法。如图 9.4 所示，两个方法画出来的图是一样的。

```

1 ggplot(economics, aes(date)) +
2   geom_line(aes(y = unemploy, colour = "unemploy")) +
3   geom_line(aes(y = uempmed, colour = "uempmed")) +
4   scale_colour_hue("variable")
5 require(reshape2)
6 emp <- melt(economics, id = "date", measure = c("unemploy", "uempmed"))
7 qplot(date, value, data = emp, geom = "line", colour = variable)

```

这些图都有一个共同的问题：两个变量取值差异太大，所以 `uempmed` 变量变成了图形底下一条平坦的线。`ggplot2` 不允许画带两个不同坐标轴的图，因为这样的图具有误导性。`ggplot2` 有两个比较直观的改进方法：把数据标度调整到相同的范围或使用自由标度的分面。这两个方法可以用下面的代码实现，结果如图 9.5 所示。

```

1 range01 <- function(x) {
2   rng <- range(x, na.rm = TRUE)
3   (x - rng[1]) / diff(rng)
4 }
5 emp2 <- ddply(emp, .(variable), transform, value = range01(value))
6 qplot(date, value, data = emp2, geom = "line",

```

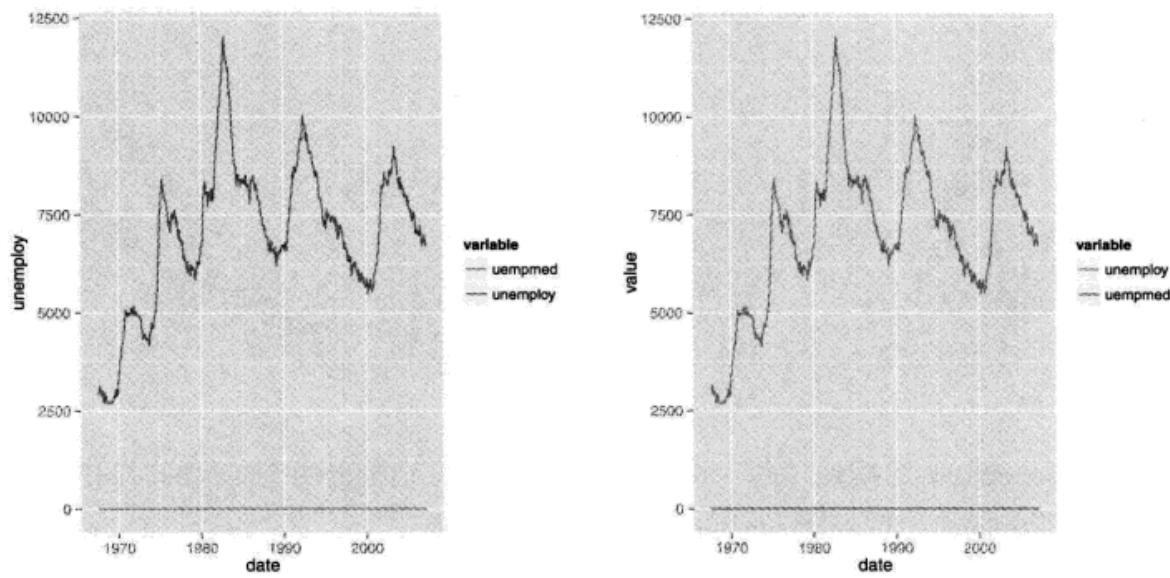


图 9.4 (参见彩图 35) 两种方法都能把两个序列画在一张图上，并且结果一样。但是“长”的数据在处理多变量的时候更方便。由于两个序列取值不在一个数量级上，所以只能看到 `unemploy` 的变化形式。仔细看会发现在图形的底下还有条线，那就是 `uempmmed` 的趋势线。

```
7 colour = variable, linetype = variable)
8 qplot(date, value, data = emp, geom = "line") +
9 facet_grid(variable ~ ., scales = "free_y")
```

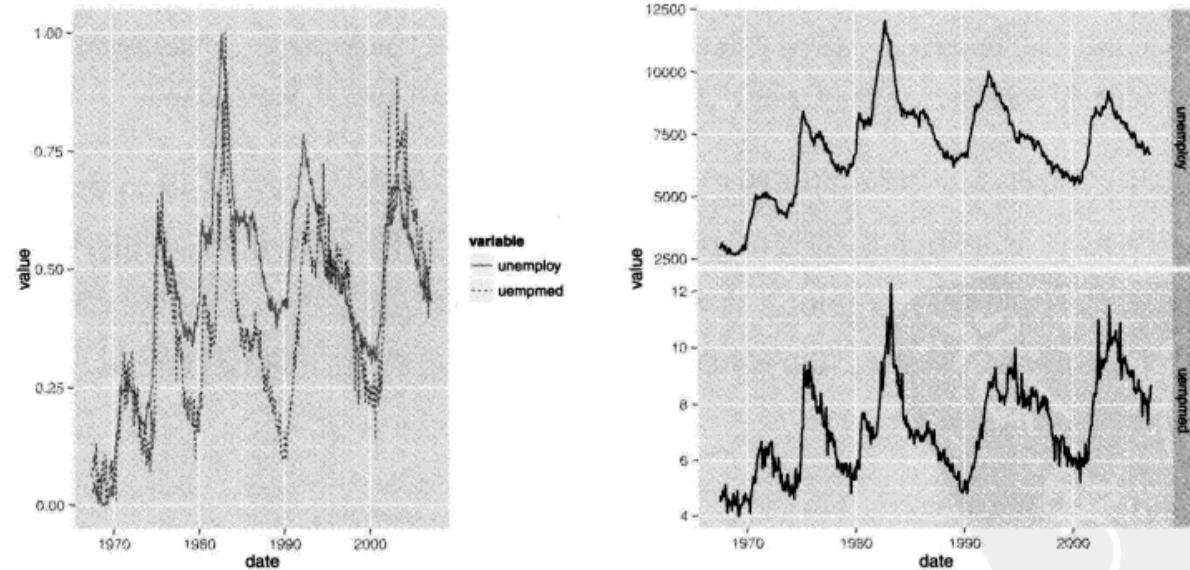


图 9.5 (参见彩图 36) 当序列数据大小差异很大时，可以使用两种方法：左图，把数据调整到相同的尺度上；右图，把数据画在不同的分面图形上。

9.2.2 平行坐标图

同样的方法能把原数据化为“长”数据，然后画出平行坐标图 (Inselberg, 1985; Wegman, 1990)。平行坐标图就是以 `variable` 变量为 x 轴表示变量名，以 `value` 为 y 轴表示变量取值。此外，还需要一个分组变量来把各个观测分组（所以每个观测分别对应一条线）。可以直接使用数据框的 `rownames` 作为分组变量，但注意不要让它跟现有变量名弄混了。我们为它取个特殊点的名字，比如 `.row`。做好了这些数据准备工作，剩下的就很容易了。

下面的平行坐标图用的是 840 部电影的用户评分数据。数据集共 10 个变量，每个变量分别对应电影在 1 到 10 分的比例。由于观测数目比较多，所以可以在这个数据上测试一下对较大数据作图的常用做法：透明度和聚类。这个数据各个变量的标度已经是统一的，所以不需要再标准化。但对于一般情形，还是要用上节的方法保证数据可比，尤其是如果还使用了其他多维方法时，更要注意这一点。

```
1 popular <- subset(movies, votes > 1e4)
2 ratings <- popular[, 7:16]
3 ratings$.row <- rownames(ratings)
4 molten <- melt(ratings, id = ".row")
```

把数据整理成这种形式后，以 `variable` 为 x 轴，以 `value` 为 y 轴，以 `.row` 分组，画出折线，就得到了平行坐标图。但是因为数值离散程度很高，所以需要在函数的默认设置上做一点小小的调整。下面的代码^② 尝试了扰动和增加透明度的方法来更好地展示数据的大致分布，如图 9.6 所示。可以看到多数电影被评 7 分或 8 分的比例都大约是 25%，只有极少数电影的 10 分比例在 35% 以上。但是这个图依然不够理想，因为大量的线条使图形的细节都看不清了，所以很难从这张图上得出很可靠的结论。

```
1 pcp <- ggplot(molten, aes(variable, value, group = .row))
2 pcp + geom_line()
3 pcp + geom_line(colour = "black", alpha = 1/20)
4 jit <- position_jitter(width = 0.25, height = 2.5)
5 pcp + geom_line(position = jit)
6 pcp + geom_line(colour = "black", alpha = 1/20, position = "jitter")
```

^②译者注：0.9.0 及之后版本中，`alpha` 参数和 `position` 参数有所修改。此处已经修正。

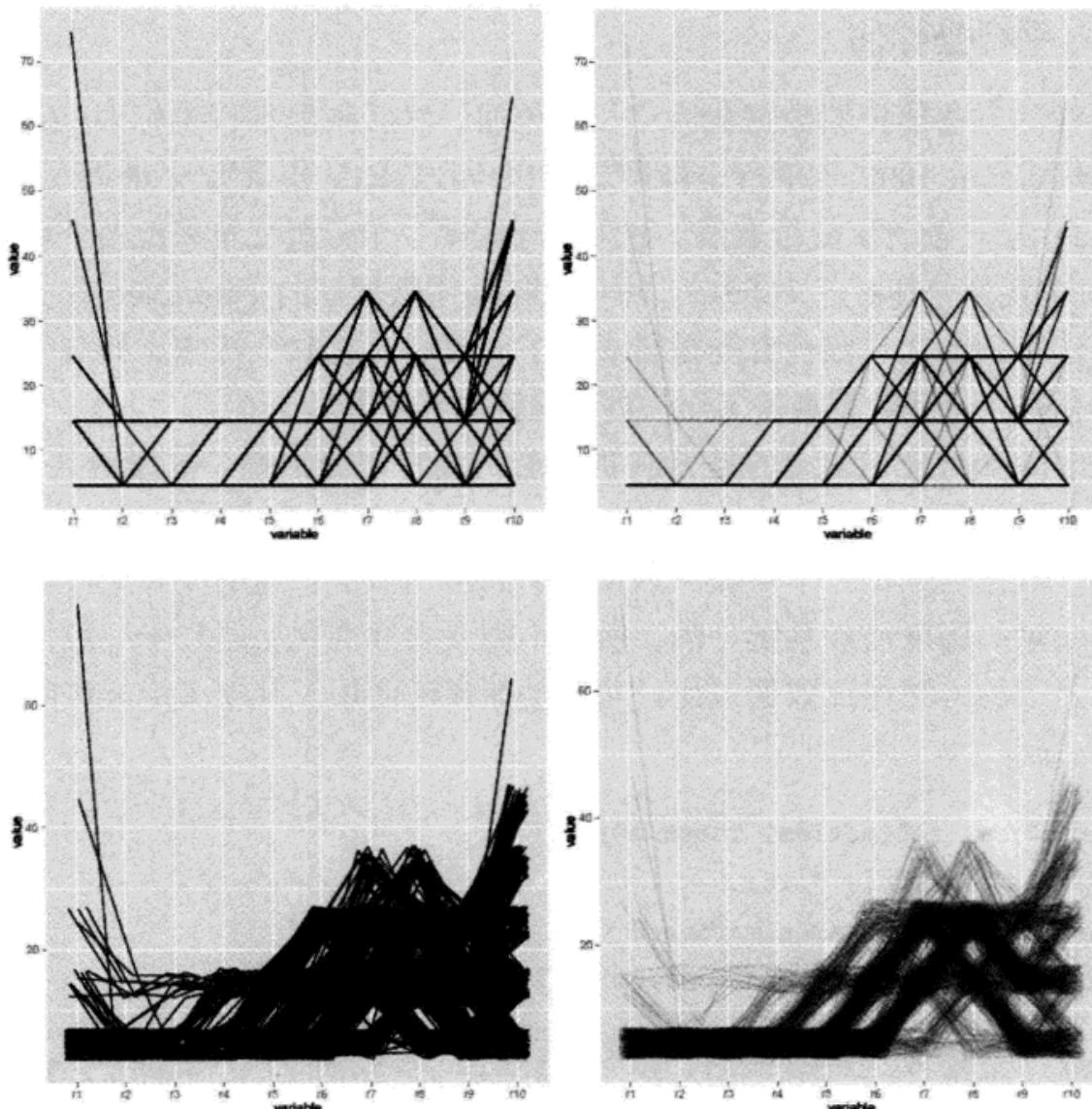


图 9.6 调整平行坐标图是更好的突出数据模式。为了改进默认的图形(左上)，我们尝试了alpha blending(右上)，jittering(左下)和同时用这两方法(右下)。

为了更清楚地观察电影得分的规律，我们把电影进行聚类，使投票模式相近的被分到一类。下面的代码用 kmeans 聚类 (Hartigan and Wong, 1979) 把所有电影分成 6 类。把平均得分最低的记为第一类，平均得分最高的记为第六类。

```

1 cl <- kmeans(ratings[1:10], 6)
2 ratings$cluster <- reorder(factor(cl$cluster), popular$rating)
3 levels(ratings$cluster) <- seq_along(levels(ratings$cluster))
4 molten <- melt(ratings, id = c(".row", "cluster"))

```

可视化聚类结果有很多种不同的方法。比较常用的做法是把不同的类标为

不同的颜色，如图 9.7 所示。然后再把每组的均值单独画在一张图上作为补充。作图代码如下^③。

```

1 pcp_cl <- ggplot(molten,
2   aes(variable, value, group = .row, colour = cluster))
3 pcp_cl + geom_line(position = "jitter", alpha = 1/5)
4 ## library(devtools)
5 ## source_gist("https://gist.github.com/4578531")
6 pcp_cl + stat_summary(aes(group = cluster), fun.y = mean,
7   geom = "line")

```

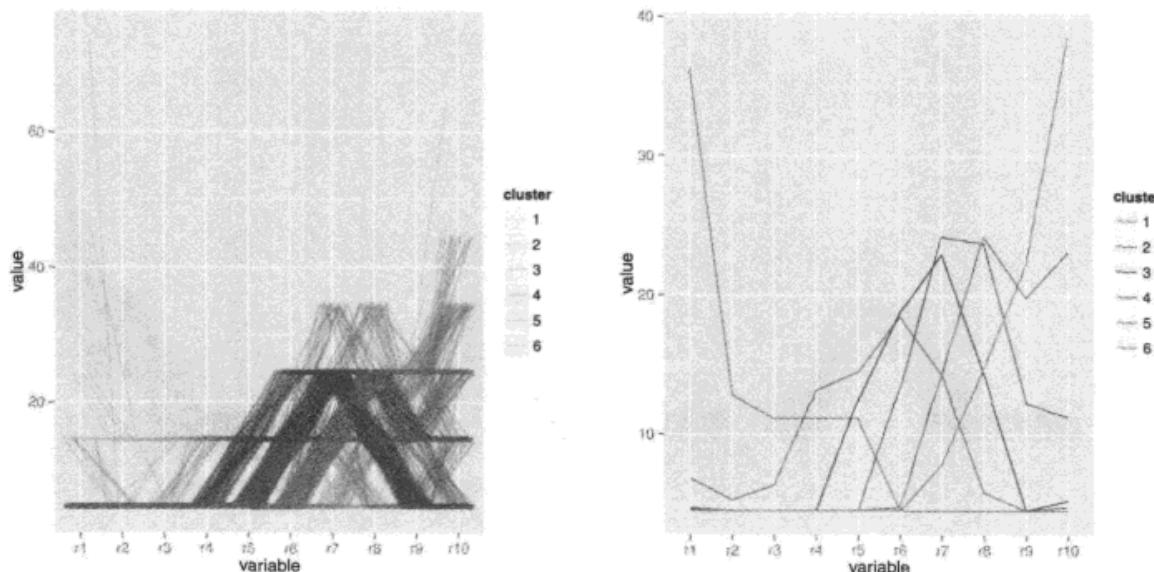


图 9.7 (参见彩图 37) 用平行坐标图展示各类数据。左图是全部数据，不同颜色表示不同类；右图是类内均值。

这些图可以很好地看出类之间的差异，但是却不能看出聚类结果好不好。

图 9.8 用 `faceting` 把每类数据画在单独的图内。这张图说明类内的差异还是很大的，说明需要增加类的个数。

```

1 pcp_cl + geom_line(position = jit, colour = "black", alpha = 1/5) +
2   facet_wrap(~ cluster)

```

^③译者注：0.9.0 及之后版本中，`alpha` 参数和 `position` 参数有所修改，此处已经修正。此外，`stat_summary` 函数在 0.9.3 版本中存在漏洞，将在下个版本修复。为正确得到第二张图，代码的两行注释中给出了 0.9.3 版本中的临时解决方案

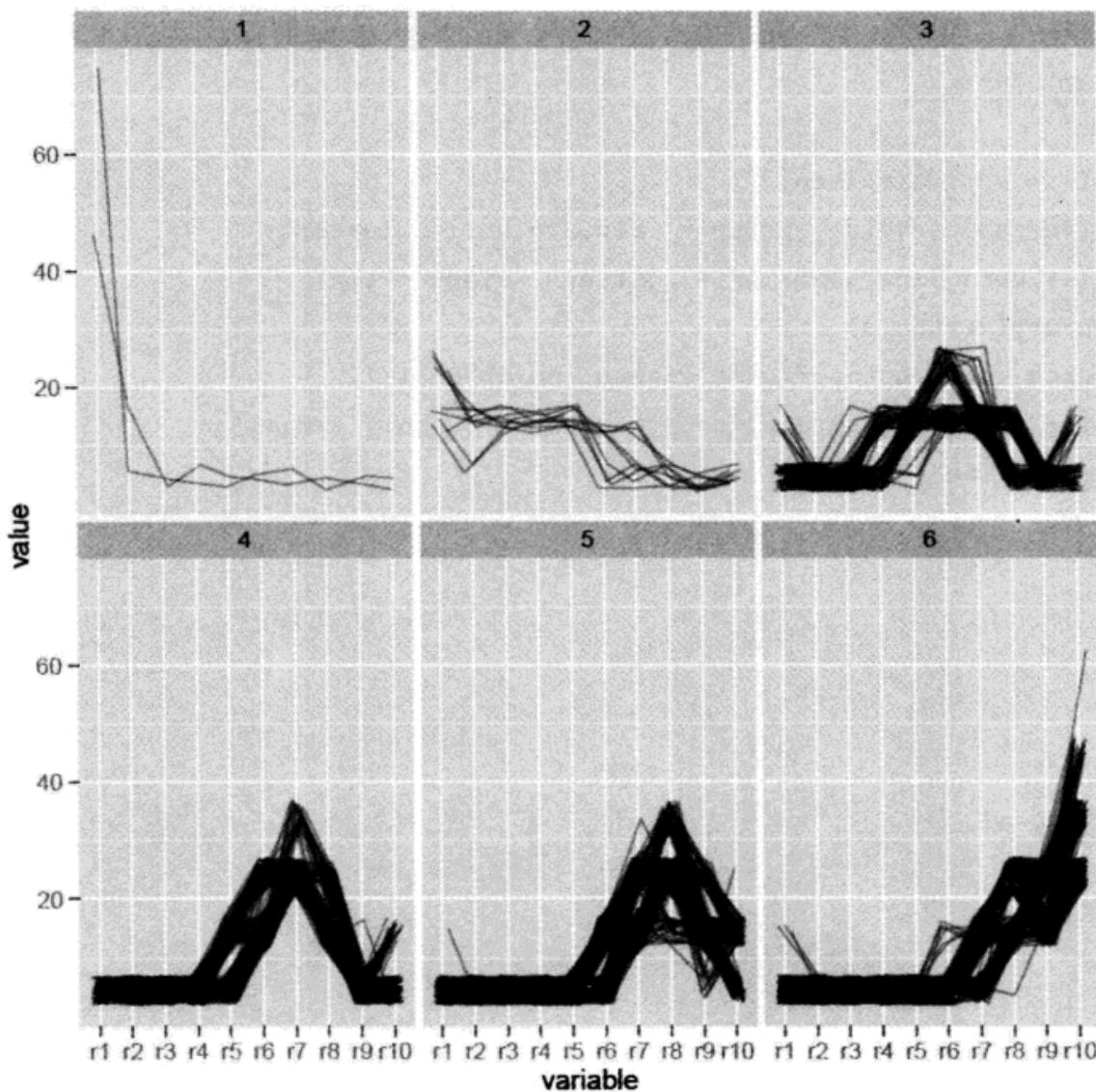


图 9.8 `faceting` 把每个类都放在单独的图内，可以看出类内差异还是相当大的，说明需要增加类的个数。

9.3 ggplot() 方法

`ggplot()` 是一个泛型函数，能根据不同的数据调用不同的方法。最常见的数据类型是数据框，所以本章一直以数据框为例。`ggplot()` 函数跟 `base` 和 `lattice` 系统一样，也可以接收其他数据类型，但是具体实现方法上有着本质的不同：`ggplot2` 并不是像它们一样直接输入数据、输出图形，而只提供作图所需要的工具。

`ggplot2` 需要借助 `fortify()` 来完成整个过程。`fortify()` 能接收一个模型或对象，以及一个可选的原始数据作为第二参数，然后把它变成能输入到

ggplot2 的形式，比如数据框。fortify 这个名称来自把模型和数据结合起来的想法：用模型预测、扩展数据，用数据修补、诊断模型，得到的结果既可以反映模型，又可以反映数据。在后面讨论线性模型增补方法时，这个概念会变得更具体一些。

本小节介绍 `fortify()` 的原理以及如何用它生成符合 ggplot2 理念的新方法。ggplot2 最重要的理念之一就是数据变形和图形展示尽可能分开进行。这样做可以让用函数的人不必局限在函数作者设想的某种特殊场合里。比如在某些情况下数据变形过程不适用了，图形展示过程还可以继续使用。

`qplot()` 只是把 `ggplot()` 进行了简单的包装，所以 `qplot()` 也可以接受不同类型的输入。

9.3.1 线性模型

ggplot2 目前只有一种适合线性模型的 `fortify` 方法。下面我们说明一下这个方法的工作原理以及如何用它画出想要的图形。图 9.9 是一个很简单的模型用 `plot.lm()` 画出的结果。这些图都是由 `plot.lm()` 预先选定好的模型诊断图。它们也许对特定的问题很有用，但是却很死板：因为如果想改动其中任何一个地方能够都必须去修改 `plot.lm()` 的源代码。可是它的源代码很难看懂，因为它的数据变形和数据展示两个中间过程纠缠在一起分不开。

而 ggplot2 把数据整理和展示完全分离开了。`fortify()` 方法负责数据变形，然后 ggplot2 负责画图。`fortify()` 把表 9.2 中由模型产生的新变量加到原数据集中。这些变量都是 `plot.lm()` 在画图中需要产生的。为了避免与数据集中的变量名冲突，我们在变量名的前面都加了“.”。

表 9.2 `fortify.lm()` 附加在模型数据上的诊断变量。

变量	描述
.cooksdi	Cook 距离
.fitted	拟合值
.hat	帽子矩阵的对角线取值
.resid	残差
.sigma	当相应的观测从模型中被剔除后残差标准差的估计值
.stdresid	标准化残差

现在我们来拟合一个很简单的模型，画出图 9.10。很显然这个模型对数据

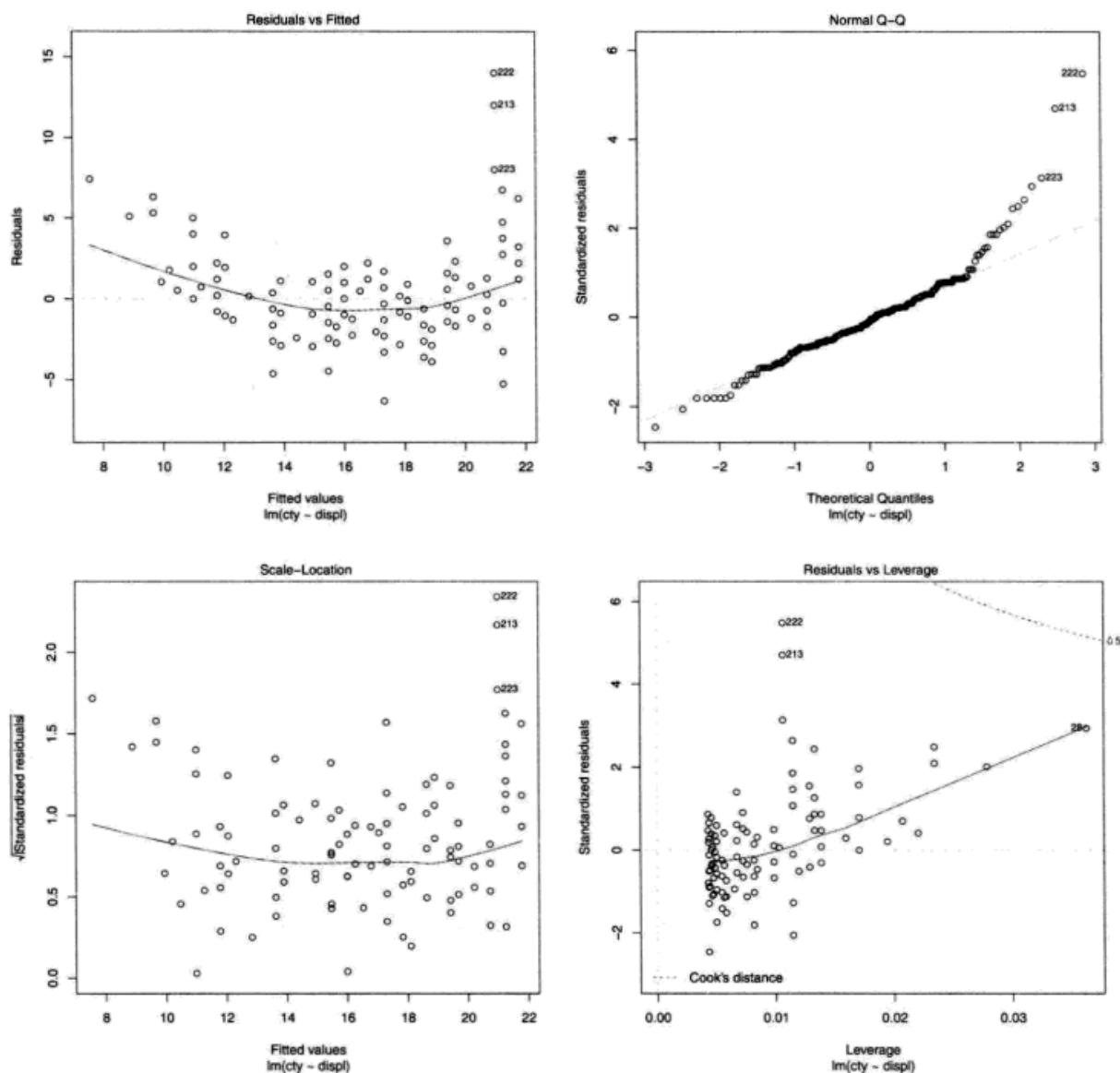


图 9.9 `plot.lm()` 一个简单模型的输出结果。

拟合得不理想，需要进行模型诊断寻找改进的方法。表 9.3 列出了 `fortify` 对这个模型的输出。我们没有提供原数据，所以它的输出结果在六个诊断变量之外自动包含了拟合模型所用的两个变量。这样就能够清晰地看到我们画图所用的数据，并且可以加入新的变量。

```
1 qplot(displ, cty, data = mpg) + geom_smooth(method = "lm")
2 mpgmod <- lm(cty ~ displ, data = mpg)
```

数据集增补后，我们可以画出比 `plot.lm()` 更美观、灵活的图，因为可以根据需要修改任何部分。下面的例子就演示了如何模仿并扩展 `plot.lm` 的第一

表 9.3 fortify(mpgmod) 的输出包含模型所用的两个变量 (cty 和 displ) 和六个诊断变量。

	cty	displ	.hat	.sigma	.cooksdi	.fitted	.resid	.stdresid
18	1.80	0.01	2.56	0.01		21.26	-3.26	-1.28
21	1.80	0.01	2.57	0.00		21.26	-0.26	-0.10
20	2.00	0.01	2.57	0.00		20.73	-0.73	-0.29
21	2.00	0.01	2.57	0.00		20.73	0.27	0.11
16	2.80	0.01	2.57	0.00		18.63	-2.63	-1.03
18	2.80	0.01	2.57	0.00		18.63	-0.63	-0.24

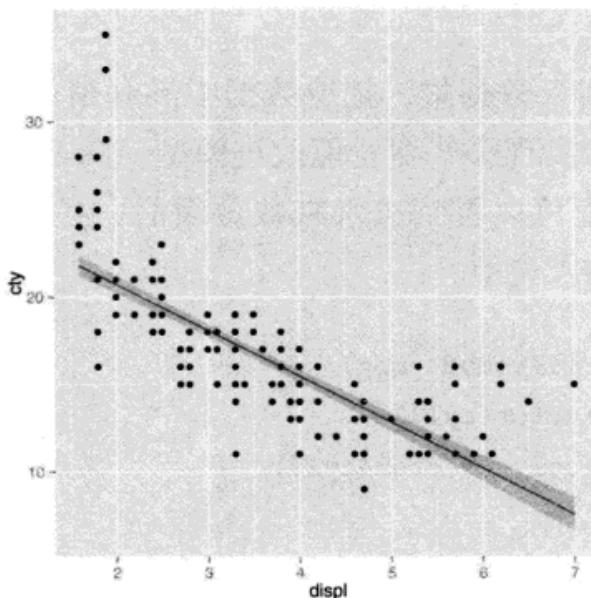


图 9.10 一个拟合得不好的简单线性模型。

幅图。在这些图的基础上，我们就可以进一步改进它：比如用标准化残差替代未标准化残差，或让点的大小代表 Cook 距离。结果如图 9.11 所示。

```

1 mod <- lm(cty ~ displ, data = mpg)
2 basic <- ggplot(mod, aes(.fitted, .resid)) +
3   geom_hline(yintercept = 0, colour = "grey50", size = 0.5) +
4   geom_point() +
5   geom_smooth(size = 0.5, se = F)
6 basic
7 basic + aes(y = .stdresid)
8 basic + aes(size = .cooksdi) + scale_size_area("Cook's distance")

```

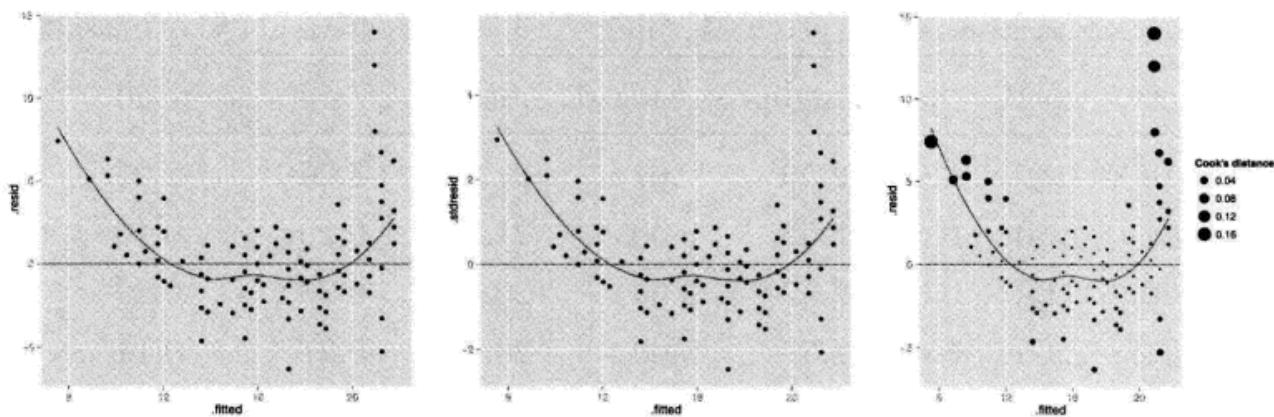


图 9.11 (左) 基本的拟合值 - 残差图。 (中) 标准化的残差。 (右) 点大小与 Cook 距离成比例。只要有全部的数据就可以很容易地修改基本图形。

我们还可以继续增补数据集，把原数据中没有被放入回归模型的变量也添加到图中。这样有助于发现哪些变量能改进模型。图 9.12 给不同汽缸数的点标上不同的颜色，从图上可以看到按汽缸数分组后，各组内的点大致呈现线性，所以需要把这个变量加入模型。

```
1 full <- basic %>% fortify(mod, mpg)
2 full + aes(colour = factor(cyl))
3 full + aes(displ, colour = factor(cyl))
```

9.3.2 编写自己的方法

编写自己的 `fortify()` 方法需要知道哪些变量对模型诊断有用以及如何把它们返回给使用者。目前对线性模型的处理方法是直接把它们加在原数据框上，但是有些时候这样做并不好。因为也许用户希望返回一个由数据框组成的列表 (`list`)，每个数据框包含不同的变量，对应不同层次的整合。

`fortify()` 并非一定要和模型返回的对象一起用。下例就是如何编写一个新的 `fortify()` 方法，把一个已有图片添加到图中。Bioconductor 上的 `EBImage` 包负责把图像导入到 R 中，`fortify()` 负责把它变成 `ggplot2` 能处理的格式 (数据框)。下面的代码把本书作者的图像添加到图中。

```
1 fortify.Image <- function(model, data, ...) {
2   colours <- channel(model, "x11")
3   colours <- colours[, rev(seq_len(ncol(colours)))]
```

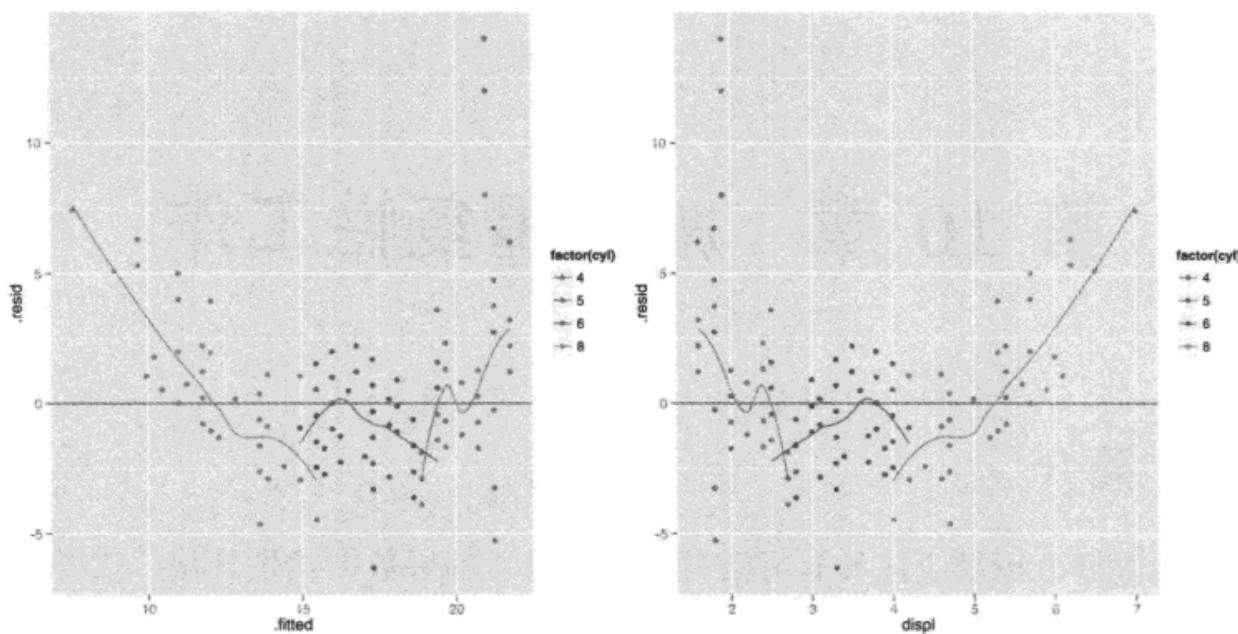


图 9.12 (参见彩图 38) 把原数据中的变量加到当前模型里可能会产生新的发现。比如 `displ` 与 `cty` 之间似乎是曲线关系, 但增加了气缸数量变量 `cyl` 后, 可以看到只要给定了 `cyl`, `displ` 与 `cty` 之间其实是呈线性关系的。

```

4   melt(colours, c("x", "y"))
5 }
6 ## 安装 EBImage 包:
7 ## source("http://bioconductor.org/biocLite.R")
8 ## biocLite("EBImage")
9
10 library(EBImage)
11 library(reshape2)
12 library(ggplot2)
13 img <- readImage("http://had.co.nz/me.jpg")
14
15 qplot(x, y, data = img, fill = value, geom="tile") +
16   scale_fill_identity() + coord_equal()

```

这个方法把数据展示和数据生成过程清晰地区分开, 显著提高了重复可用性, 但却没有提供任何封装在一起的函数。用户如果想要生成线性模型的诊断图, 需要自己把各个部分组装在一起。只要准备好了可以查看并修改的基本框架, 就可以写个函数把所有部分合成一块, 生成想要的图形。更多细节请参见 10.4 节。

第 10 章 减少重复性工作

10.1 简介

好的数据分析都应该具有灵活性这一优点。如果数据发生变化，或者出现一些很不利于基本假设的信息时，这时候我们应该能够快速、便捷地更改之前的图形。灵活性的主要敌人是重复。如果你的代码中很多都是重复、冗余的，那么当发生变化时就需要一处一处地去修改。这些繁杂的修改工作往往是很令人抓狂、沮丧的。

本章主要介绍三种减少重复性工作的方法。在 10.2 节中，介绍如何在之前图形的基础上迭代地进行修改，而不是重新输入很多重复性的代码。在 10.3 节中，介绍如何制作具有重复处理机制，并且只需定义一次就可以多次使用的“绘图模板”。在 10.4 节中，介绍如何来写可以绘制、修改图形的函数。

10.2 迭代

`ggplot2` 会将你最后一次绘制或者修改的图形储存下来，输入 `last_plot()` 即可获取该图形。我们可以从一个最基本的图形开始，迭代式地逐步添加图层和调整标度，直到得到最终有意思的结果——这种交互式的工作往往是很有用的。下面的代码展示了将一个我们感兴趣的图形区域逐步放大，并且添加新的图层来突出显示我们发现的有趣信息： x 值和 y 值相同的钻石很少。参见图 10.1。

```
1 qplot(x, y, data = diamonds, na.rm = TRUE)
2 last_plot() + xlim(3, 11) + ylim(3, 11)
3 last_plot() + xlim(4, 10) + ylim(4, 10)
4 last_plot() + xlim(4, 5) + ylim(4, 5)
```

```
5 last_plot() + xlim(4, 4.5) + ylim(4, 4.5)  
6 last_plot() + geom_abline(colour = "red")
```

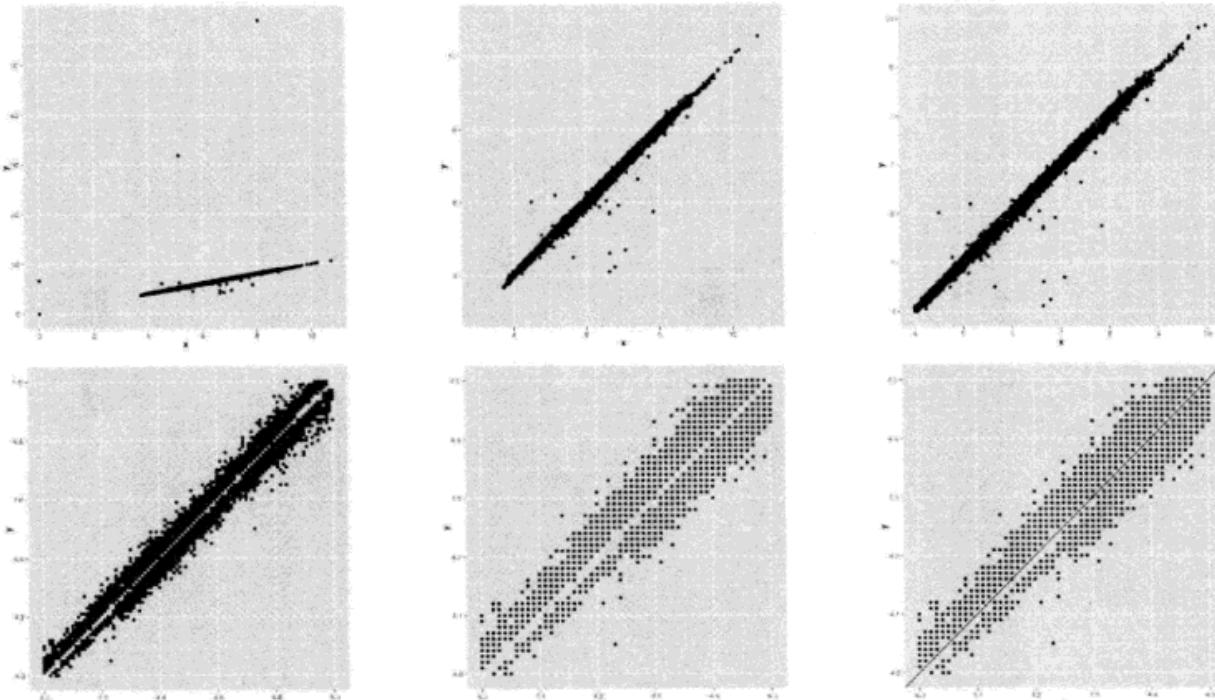


图 10.1 当不断放大图形时，交互地使用 `last_plot()` 可以快速有效地找到最佳视图。最后一张图上添加了一条截距为 0 斜率为 1 的直线。图中可以看到数据中没有正方形的钻石。

将图形调整至所期望的样子后，最好创建一个能生成最终图形的独立代码，这样好处在于可以不依赖于之前的步骤；此外，或许还应该添加必要的注释来说明如此绘图的原因。下面是上文中经过交互式修改后得到最终图形的代码。

```
1 qplot(x, y, data = diamonds, na.rm = T) +  
2   geom_abline(colour = "red") +  
3   xlim(4, 4.5) + ylim(4, 4.5)
```

10.3 绘图模板

`ggplot2` 图形的每一个组件都是一个对象：可以被创建、存储并独立应用于某个图形中。鉴于此，我们可以创建可重用的组件来自动执行某些常用的任务，从而不用多次重复输入冗长的函数。下面的例子创建了一个颜色标度并将其应用在两个图形中，参见图 10.2。

```

1 gradient_rb <- scale_colour_gradient(low = "red", high = "blue")
2 qplot(cty, hwy, data = mpg, colour = displ) + gradient_rb
3 qplot(bodywt, brainwt, data = msleep, colour = awake, log = "xy") +
4   gradient_rb

```

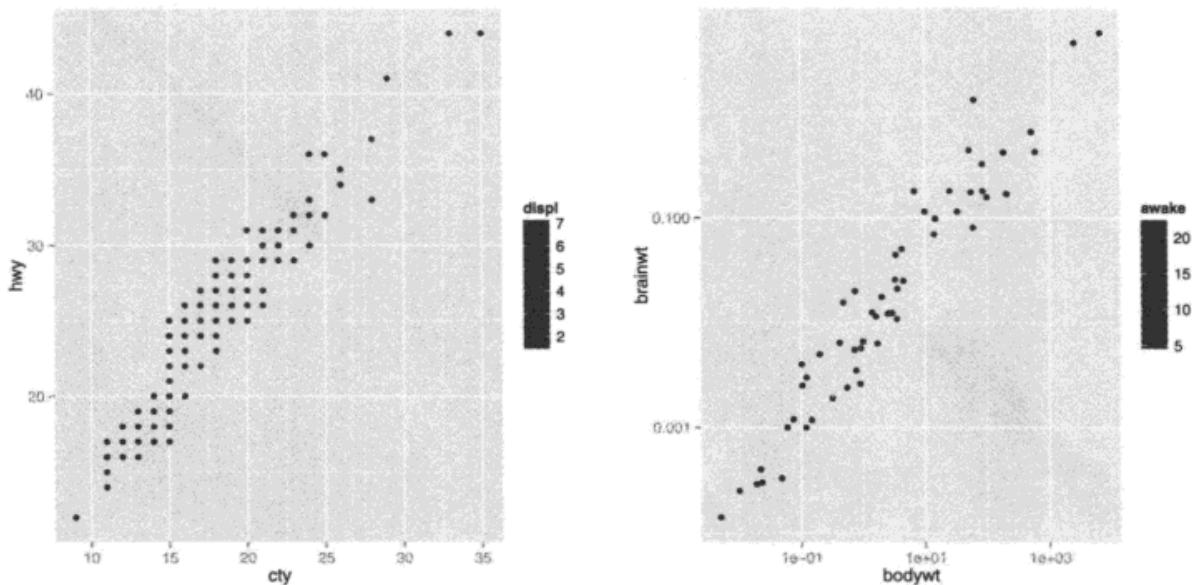


图 10.2 (参见彩图 39) 将标度存储于某一变量中可供很多图形便捷地调用。对图层和分面也可进行类似地操作。

同存储单个对象一样，也可以将 ggplot2 中的组件存储为 list 格式的列表。向某个图形中添加组件列表和将其中的组件按顺序逐个添加是一样的效果。下面的例子创建了两个连续标度，用来取消坐标轴标签和刻度。这些对象只需要创建一次，便可应用于很多不同的图形中，代码如下，结果如图 10.3 所示。

```

1 ## 原书中的 NA 替换为了 NULL，向量 c() 换成了列表 list()
2 xquiet <- scale_x_continuous("", breaks = NULL)
3 yquiet <- scale_y_continuous("", breaks = NULL)
4 quiet <- list(xquiet, yquiet)
5
6 qplot(mpg, wt, data = mtcars) + quiet
7 qplot(displ, cty, data = mpg) + quiet

```

类似地，创建一个改变图层默认设置的函数也很简单。例如，如果想要创建一个在图形中添加线性模型的函数，代码如下，结果如图 10.4 所示。

```

1 geom_lm <- function(formula=y~x){
2   geom_smooth(formula = formula, se = FALSE, method = "lm")
}

```

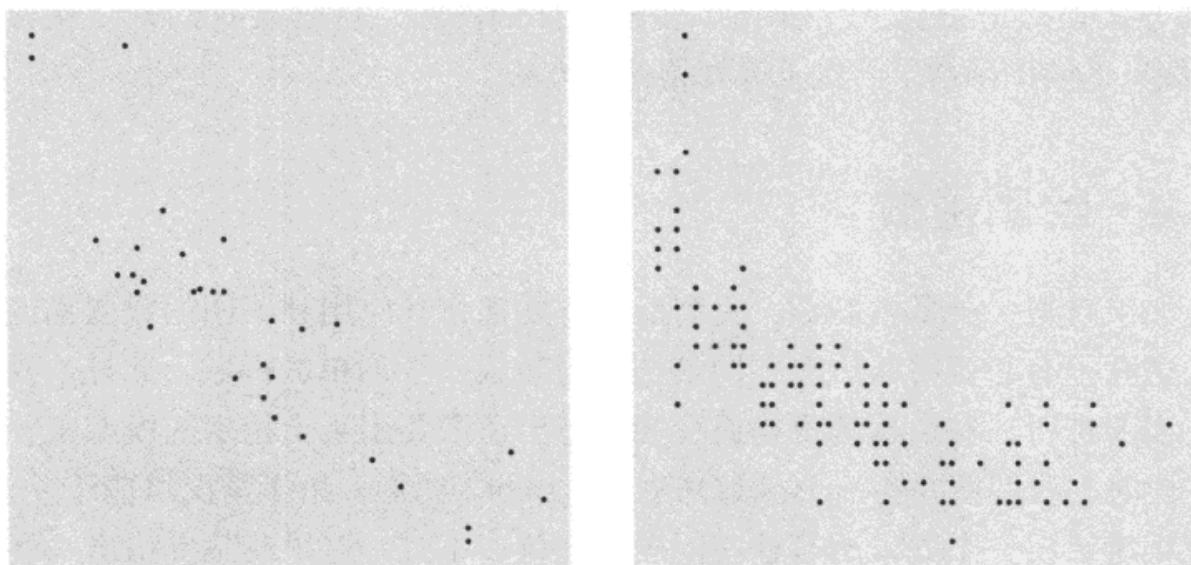


图 10.3 使用“静默”的 x 和 y 标度时，移除了标签并隐藏了刻度和网格线。

```
3 }
4 qplot(mpg, wt, data = mtcars) + geom_lm()
5 library(splines)
6 qplot(mpg, wt, data = mtcars) + geom_lm(y ~ ns(x, 3))
```

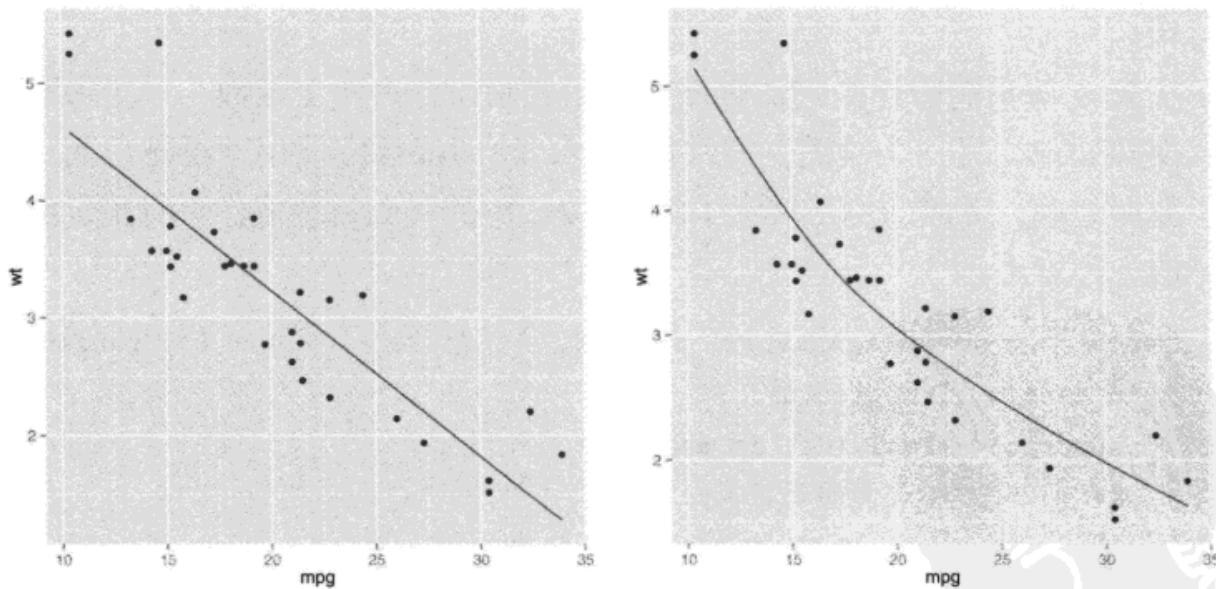


图 10.4 创建一个自定义的几何对象函数，用类似（但不相同）的组件创建图形时即可调用。

我们定义的函数可以更为复杂，比如在一个列表中包含很多个组件。通过定义此类函数的方式，能够很便捷地画出很复杂的图形。如果我们想用一种特

定的方式便捷地创建一个糅合了多种组件的图形，这时候我们就需要写一个“高级”的绘图函数了。接下来介绍这种方法。

10.4 绘图函数

如果对某一种基本图形，我们反反复复地将其应用到不同的数据集或参数上，这时很有必要将其所有不同的选项封装成一个简单的函数。此外，我们可能还需要进行一些数据的重构或变换，甚至还需要用某个特定的模型来分析数据。此时，我们需要写一个函数来生成 `ggplot2` 图形。由于情况可能千差万别，因此很难给出一个处理所有这些问题的建议；不过，这里还是要指出一些很重要，并且必须要考虑的问题。

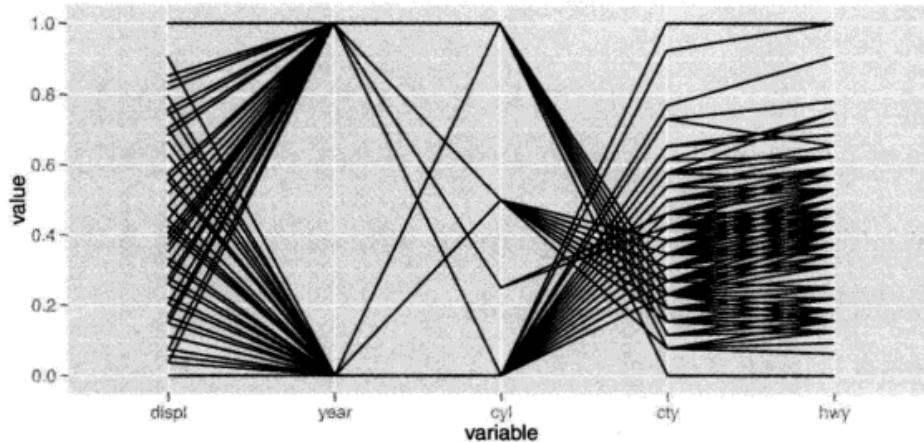
- 由于是在函数的环境中创建图形，因此以数据框的格式向 `ggplot()` 传递参数时需要格外小心。此外，还要确保没有在图形属性映射中误用了任何函数的局部变量。
- 若允许用户向图形属性映射中提供自定义变量，建议使用 `aes_string()`。该函数作用与 `aes()` 类似，但它使用字符串而不是未估值的表达式 (unevaluated expressions)。`aes_string("cty", colour = "hwy")` 与 `aes(cty, colour = hwy)` 的功能是一样的，但字符串要比表达式更容易使用。
- 在第 9 章我们曾经提到，应该把绘图代码切分为两个函数，一个做数据变换和处理，另一个绘图。通常情况下，绘图函数应仅用于绘图，而不要做数据处理。下面的例子展示了将 § 9.2.2 中的代码封装成作平行坐标图函数的一种方法。

```
1 ## 译者注：请加载 reshape2、plyr 包，并运行 9.2 节中自定义的 range01() 函数
2 pcp_data <- function(df) {
3   numeric <- lapply(df, is.numeric)
4   ## 每一列的数值调整到相同的范围
5   df[numeric] <- colwise(range01)(df[numeric])
6   ## 行名作为行识别信息
7   df$.row <- rownames(df)
8   ## Melt 将非数值变量作为 id.vars
9   dfm <- melt(df, id = c(".row", names(df)[!numeric]))
10  ## 给数据框添加 pcp 类
```

```

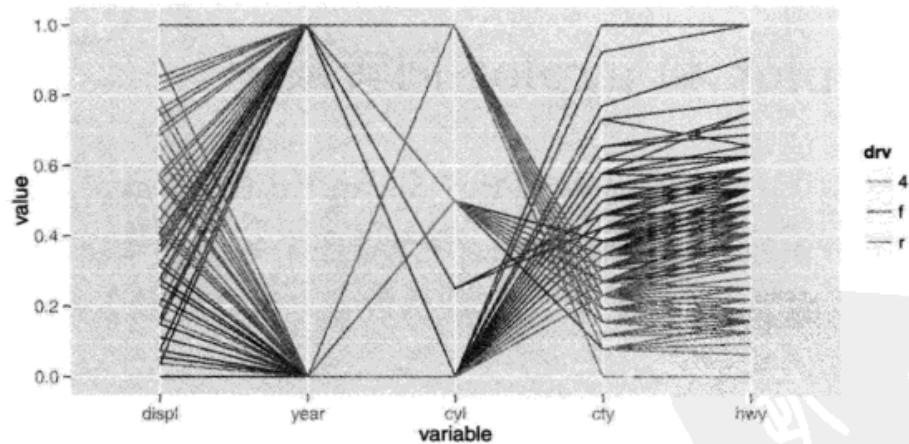
11   class(dfm) <- c("pcp", class(dfm))
12   dfm
13 }
14 pcp <- function(df, ...) {
15   df <- pcp_data(df)
16   ggplot(df, aes(variable, value)) + geom_line(aes(group = .row))
17 }
1 > pcp(mpg) ## 绘制平行坐标图:

```



绘制平行坐标图并以不同颜色来区分类别 (以 `drv` 划分), 参见彩图 40:

```
1 > pcp(mpg) + aes(colour = drv)
```



封装函数的一个很好的例子是 `qplot()`, 如果读者想自己写类似的函数, 笔者强烈建议逐行认真阅读 `qplot()` 函数的源代码, 去理解它的工作机理。如果你能以此方式深入学习本书的话, 那么你一定会熟练掌握 `ggplot2` 所有相关的代码: 图海无涯 R 作舟, 统计图形的种类其实很多, 困难是如何用 R 语言去正确地构思并实现。

附录 A 不同语法间的转换

A.1 简介

`ggplot2` 不是孤立存在的，它是 R 及其他软件的图形工具的发展中的一部分。这一章将介绍如何在 `ggplot2` 命令和其他绘图系统之间进行转换。

- `ggplot2` 中，`qplot()` 和 `ggplot()` 间的语法，见 § A.2；
- 从基础图形系统转换，见 § A.3；
- 从 `lattice` 图形系统转换，见 § A.4；
- 从 GPL(Graphics Production Library) 转换，见 § A.5。

每个部分会给出一些例子，来大致介绍如何在不同类型间进行转换。

A.2 在 `qplot` 和 `ggplot` 间转换

在 `ggplot2` 中，有两种基本绘图方式，`qplot()` 和 `ggplot()`。`qplot()` 可以在很多情况下快速、便捷地使用；但当用多个数据源来设计多个图层的图形时，它就捉襟见肘了。本节将介绍其默认设置，以及它是怎样映射到 `ggplot()` 语法的。

默认设置下 `qplot()` 假定你想要一幅散点图，也就是使用 `geom_point()` 函数。

```
1 qplot(x, y, data = data)
2 ggplot(data, aes(x, y)) + geom_point()
```

A.2.1 图形属性

如果需要添加其他图形属性，则它们会被添加在默认设置之上。qplot()中，不同的图层之间不能使用不同的图形属性映射或数据源。

```
1 qplot(x, y, data = data, shape = shape, colour = colour)
2 ggplot(data, aes(x, y, shape = shape, colour = colour)) + geom_point()
```

qplot() 中的图形属性参数总是试图将图形属性映射到一个变量上。当参数不是一个变量而是其他值时，该值将会作为新的一列加在原数据集上。如果不想用默认情况，而是要设置某一个图形属性到一个特定值，那么可以在 qplot() 中用 I() 将该值括住，或将其传递为图层的一个参数。§ 4.5.2 详细描述了设置和映射的不同。

```
1 qplot(x, y, data = data, colour = I("red"))
2 ggplot(data, aes(x, y)) + geom_point(colour = "red")
```

A.2.2 图层

改变几何对象参数可以改变添加到图形上的几何对象。

```
1 qplot(x, y, data = data, geom = "line")
2 ggplot(data, aes(x, y)) + geom_line()
```

如果给几何对象参数传递了一个包含了多个几何对象名称的向量，那么每个几何对象将会按次序添加。

```
1 qplot(x, y, data = data, geom = c("point", "smooth"))
2 ggplot(data, aes(x, y)) + geom_point() + geom_smooth()
```

和 ggplot2 中的其他部分不同，统计变换和几何对象是独立的。

```
1 qplot(x, y, data = data, stat = "bin")
2 ggplot(data, aes(x, y)) + geom_point(stat = "bin")
```

任意图层的参数都会传递给所有图层。大部分图层都会忽略对其无用的参数。

```
1 qplot(x, y, data = data, geom = c("point", "smooth"), method = "lm")
2 ggplot(data, aes(x, y)) + geom_point(method = "lm") + geom_smooth(method = "lm")
```

A.2.3 标度和坐标轴

我们可以用 `xlim`, `ylim`, `xlab` 及 `ylab` 来控制 x 轴和 y 轴的基本属性。

```

1 qplot(x, y, data = data, xlim = c(1, 5), xlab = "my label")
2 ggplot(data, aes(x, y)) + geom_point() +
3   scale_x_continuous("my label", limits = c(1, 5))
4
5 qplot(x, y, data = data, xlim = c(1, 5), ylim = c(10, 20))
6 ggplot(data, aes(x, y)) + geom_point() +
7   scale_x_continuous(limits = c(1, 5))
8   scale_y_continuous(limits = c(10, 20))

```

同 `plot()` 一样, `qplot()` 可以很方便地对坐标轴进行指数变换。不过有很多变换在 `qplot()` 是办不到的, 更多细节参见 § 6.4.2。

```

1 qplot(x, y, data = data, log="xy")
2 ggplot(data, aes(x, y)) + geom_point() +
3   scale_x_log10() + scale_y_log10()

```

A.2.4 绘图选项

`qplot()` 可以识别很多绘图选项 (和 `plot()` 选项类似), 并将它们转换成 `ggplot2` 适用的形式。§ 8.1.2 列出了所有可能的绘图选项及其作用。

```

1 qplot(x, y, data = data, main = "title", asp = 1)
2 ggplot(data, aes(x, y)) + geom_point() +
3   labs(title = "title") + coord_fixed(ratio = 1)

```

A.3 基础图形系统

基础图形系统中有两类绘图函数, 一种是绘制完整的图形, 另一种是在现有图形上添加元素; 专有名词分别是高级绘图和低级绘图。

A.3.1 高级绘图

`qplot()` 是模仿 `plot()` 设计的，可以完成绝大多数高水平绘图命令的工作。基础图形中只有两种图是 `ggplot2` 不能代替的：`filled.contour()` 和 `persp()`。

```
1 plot(x, y); dotchart(x, y); stripchart(x, y)
2 qplot(x, y)
3
4 plot(x, y, type = "l")
5 qplot(x, y, geom = "line")
6
7 plot(x, y, type = "s")
8 qplot(x, y, geom = "step")
9
10 plot(x, y, type = "b")
11 qplot(x, y, geom = c("point", "line"))
12
13 boxplot(x, y)
14 qplot(x, y, geom = "boxplot")
15
16 hist(x)
17 qplot(x, geom = "histogram")
18
19 cdplot(x, y)
20 qplot(x, fill = y, geom = "density", position = "fill")
21
22 coplot(y ~ x | a + b)
23 qplot(x, y, facets = a ~ b)
```

这些几何对象的参数和基础图形是不同的。例如，`hist()` 的参数是箱子的个数，而 `geom_histogram()` 的参数则是组距。

```
1 hist(x, bins = 100)
2 qplot(x, geom = "histogram", binwidth = 1)
```

`qplot()` 的数据格式要求和基础图形也时常有所不同。例如，条形图 (`bar geom`) 可适用于没有汇总的数据^①，而 `barplot()` 则适用于汇总后的数据。`tile`

^①译者注：这里的汇总是指用 `table()` 等函数对数据进行汇总统计。

和 `contour` 这两个几何对象要求数据框形式的参数，而 `image()` 和 `contour()` 则需要矩阵形式的参数。

```

1 barplot(table(x))
2 qplot(x, geom = "bar")
3
4 barplot(x)
5 qplot(names(x), x, geom = "bar", stat = "identity")
6
7 image(x)
8 qplot(X1, X2, data = melt(x), geom = "tile", fill = value)
9
10 contour(x)
11 qplot(X1, X2, data = melt(x), geom = "contour", fill = value)

```

通常情况下，基础绘图函数使用独立的向量，而不像 `ggplot2` 使用整合好的数据框。当数据未被指定时，`qplot` 会尝试构建一个数据框，但不能保证总能实现。如果有奇怪的错误产生时，你可能需要自己构建数据框。

```

1 with(df, plot(x, y))
2 qplot(x, y, data = df)

```

默认时，`qplot()` 将以某一标度将数值映射到图形属性中。若需要自行设置图形属性并覆盖默认值，则需使用 `I()`。

```

1 plot(x, y, col = "red", cex = 1)
2 qplot(x, y, colour = I("red"), size = I(1))

```

A.3.2 低级绘图

低级绘图函数将在已有图形上进行添加，这和 `ggplot2` 中添加一个新的图层一样，参见表 A.1。

```

1 plot(x, y)
2 lines(x, y)
3
4 qplot(x, y) + geom_line()
5

```

表 A.1 基础图形中的低级绘图函数和 ggplot2 中图层的对比。

基础函数	ggplot2 的图层
curve()	geom_curve()
hline()	geom_hline()
lines()	geom_line()
points()	geom_point()
polygon()	geom_polygon()
rect()	geom_rect()
rug()	geom_rug()
segments()	geom_segment()
text()	geom_text()
vline()	geom_vline()
abline(lm(y ~ x))	geom_smooth(method = "lm")
lines(density(x))	geom_density()
lines(loess(x, y))	geom_smooth()

```

6 ## 或者:
7 qplot(x, y)
8 last_plot() + geom_line()

```

A.3.3 图例、坐标轴和网格线

在 ggplot2 中，图例和坐标轴是由相关的标度控制的。坐标轴由 `x` 和 `y` 标度生成，其他的标度来生成图例。具体内容参见图形主题部分：§ 8.1 介绍改变坐标轴和图例的外观；§ 6.5 介绍如何改变其内容。网格线的展现由 `grid.major` 和 `grid.minor` 主题选项控制，它们的位置由 `x` 和 `y` 标度确定。

A.3.4 调色板

ggplot2 每个独立的图形都有各自的标度，而不是使用全局调色板。大多数情况下，可以使用默认的颜色标度（一般都配置了较好的视觉属性）；如果想重复使用某个现存调色板时，可以使用 `scale_colour_manual()` 来实现。但要格外注意的是，一定要为所绘制的图形选择恰当的颜色！

```

1 palette(rainbow(5))
2 plot(1:5, 1:5, col = 1:5, pch = 19, cex = 4)
3
4 qplot(1:5, 1:5, col = factor(1:5), size = I(4))
5 last_plot() + scale_colour_manual(values = rainbow(5))

```

在 ggplot2 中，你还可以使用连续色彩的调色板，其中的色彩是由线性插值得到的。

```

1 qplot(0:100, 0:100, col = 0:100, size = I(4)) +
2   scale_colour_gradientn(colours = rainbow(7))
3 last_plot() +
4   scale_colour_gradientn(colours = terrain.colors(7))

```

A.3.5 绘图参数

基础图形中 par 函数的大部分选项和 ggplot2 的主题系统类似，或者是几何对象和标度的默认参数。用 box() 绘制的图形边界可以类似地由 panel.background 和 plot.background 主题元素控制。此外，ggplot2 图形的标题由 title 选项控制，而不是 title() 函数。

A.4 lattice 图形设备

lattice 和 ggplot2 的主要区别在于，lattice 的图形公式是基于公式的。而 ggplot2 并非如此，这是因为公式化不能很好地推广到更复杂的情况下。

```

1 xyplot(rating ~ year, data = movies)
2 qplot(year, rating, data = movies)
3
4 xyplot(rating ~ year | Comedy + Action, data = movies)
5 qplot(year, rating, data = movies, facets = ~ Comedy + Action)
6 ## 或者：
7 qplot(year, rating, data = movies, facets = Comedy ~ Action)

```

lattice 中有很多不同的函数来生成不同类型的图形（如同面板参数中的设置），而 ggplot2 有 qplot()。

```

1 stripplot(~ rating, data = movies, jitter.data = TRUE)
2 qplot(rating, 1, data = movies, geom = "jitter")
3
4 histogram(~ rating, data = movies)
5 qplot(rating, data = movies, geom = "histogram")
6
7 bwplot(Comedy ~ rating ,data = movies)
8 qplot(factor(Comedy), rating, data = movies, type = "boxplot")
9
10 xyplot(wt ~ mpg, mtcars, type = c("p", "smooth"))
11 qplot(mpg, wt, data = mtcars, geom = c("point", "smooth"))
12
13 xyplot(wt ~ mpg, mtcars, type = c("p", "r"))
14 qplot(mpg, wt, data = mtcars, geom = c("point", "smooth"),
15   method = "lm")

```

ggplot2 和 lattice 的标度处理方式相似，只是在语法上有些不同。

```

1 xyplot(wt ~ mpg | cyl, mtcars, scales = list(y = list(relation= "free")))
2 qplot(mpg, wt, data = mtcars) + facet_wrap(~ cyl, scales = "free")
3
4 xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 10))
5 qplot(mpg, wt, data = mtcars, log = "xy")
6
7 xyplot(wt ~ mpg | cyl, mtcars, scales = list(log = 2))
8 qplot(mpg, wt, data = mtcars) + scale_x_log2() + scale_y_log2()
9
10 xyplot(wt ~ mpg, mtcars, group = cyl, auto.key = TRUE)
11 ## 直接将图形属性映射到颜色、大小、形状等
12 qplot(mpg, wt, data = mtcars, colour = cyl)
13
14 xyplot(wt ~ mpg, mtcars, xlim = c(20,30))
15 qplot(mpg, wt, data = mtcars, xlim = c(20,30))

```

lattice 和 ggplot2 中都有类似的选项来控制图形中的标签。

```

1 xyplot(wt ~ mpg, mtcars,
2   xlab = "Miles per gallon", ylab = "Weight",
3   main = "Weight-efficiency tradeoff")

```

```

4 qplot(mpg, wt, data = mtcars,
5   xlab = "Miles per gallon", ylab = "Weight",
6   main = "Weight-efficiency tradeoff")
7
8 xyplot(wt ~ mpg, mtcars, aspect = 1)
9 qplot(mpg, wt, data = mtcars, asp = 1)

```

`par.settings()` 等同于 `+ theme()`, `trellis.options.set()` 等同于 `theme_set`,
`trellis.par.get()` 等同于 `theme_get()`。

更复杂的 `lattice` 公式的效果类似于数据重排后再使用 `ggplot2`。

A.5 GPL

《The Grammar of Graphics》(后文简写为 GoG) 包括两套设置。一套格式非常简明，用来标注图形，另一套比较复杂，以 `xml` 格式将图形存储在硬盘上。下面是来自 Wilkinson 书中的简明格式的一个例子(第 13 页图 1.5)。

```

1 DATA: source("demographics")
2 DATA: longitude, latitude = map(source("World"))
3 TRANS: bd = max(birth - death, 0)
4 COORD: project.mercator()
5 ELEMENT: point(position(lon * lat), size(bd), color(color.red))
6 ELEMENT: polygon(position(longitude * latitude))

```

下面以相对简单的形式将其改编成 `ggplot2` 的语法：

- `ggplot()` 用来指定默认数据和默认图形属性的映射。
- 数据在全局环境下由 R 数据框提供，不需要明确地进行加载。我们也用了另一种格式稍微不同的世界地图数据集，其中的列分别是经度和纬度；这样我们可以对这两个数据集都用同样的图形属性映射。后加入的图层可以覆盖图形中默认使用的数据和图形属性映射关系。
- 用相应的 R 代码替换第三行的 `TRANS`。
- 两个 `ELEMENT` 由相应图层(需明确指定数据源)进行替换。每个几何对象都有一个默认的统计变换，在绘图前对数据进行变换。在这个例子的几何对象中，默认的统计变换是恒等函数。修正的图形属性(这个例子中的红色)作为附加参数提供给了图层，而不是作为一个特殊的常数。

- 这个例子里省略了 SCALE 部分(使用默认值)。在 ggplot2 和 GoG 这两个例子中，标度都是默认设定的。在 ggplot 中，你可以添加一个标度对象来覆盖默认设置，比如用 scale_colour 或 scale_size。
- COORD 使用了一个稍有不同的格式。通常情况下，为了方便 R 用户，ggplot 中组件的定义和 GoG 中的会有些差异。
- 用 + 号运算将所有的部分组合到一起，便可得到最后的图形。

综上所述，等价的 ggplot2 代码如下：

```
1 demographics <- transform(demographics, bd = pmax(birth - death, 0))
2
3 ggplot(demographic, aes(lon, lat)) +
4   geom_polygon(data = world) +
5   geom_point(aes(size = bd), colour = "red") +
6   coord_map(projection = "mercator")
```

附录 B 图形属性的定义

本章附录对 `grid` 绘图函数中用到的各种格式进行了总结。这部分的内容大多零散地分布在 R 的帮助文档中，这里将这些信息进行了汇总。

B.1 颜色

颜色可按如下方式定义：

- 用名字指定，比如"red"。这些颜色如图 B.1(a) 所示，可以用 `colours()` 得到详细的颜色列表。Stowers Institute 提供了一份可打印的 pdf 文件，其列出了所有颜色：<http://research.stowers-institute.org/efg/R/Color/Chart/>。
- 用 RGB 方式指定，形如 "#RRGGBB"，其中每对 RR, GG 和 BB 都是从 00 到 FF 的一个十六进制数字。我们还可以用 `alpha()` 来指定部分的透明度，比如 `alpha("red", 0.5)`。
- 用 NA 指定，表示完全透明色。

函数 `rgb()`, `hsv()` 和 `hcl()` 可在不同的色彩空间中创建颜色。

B.2 线条类型

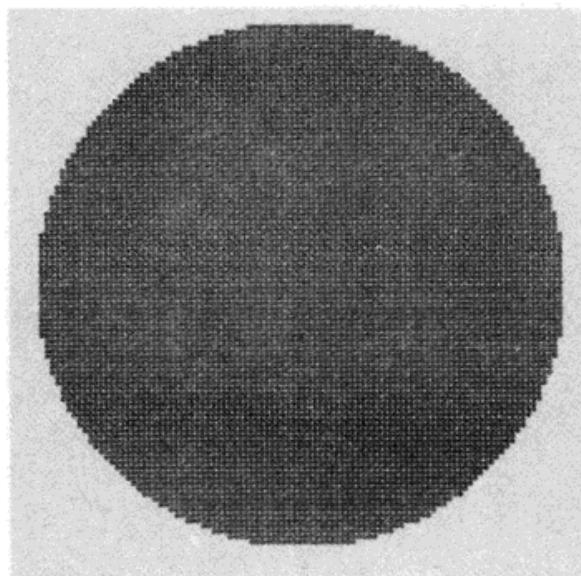
线条类型可按如下方式定义：

- 用整数或名字指定：0 = `blank`, 1 = `solid`, 2 = `dashed`, 3 = `dotted`, 4 = `dotdash`, 5 = `longdash`, 6 = `twodash`，如图 B.1(b) 所示。
- 线条的“开/关”长度模式。由偶数个（最多 8 个）16 进制的数字合起来构

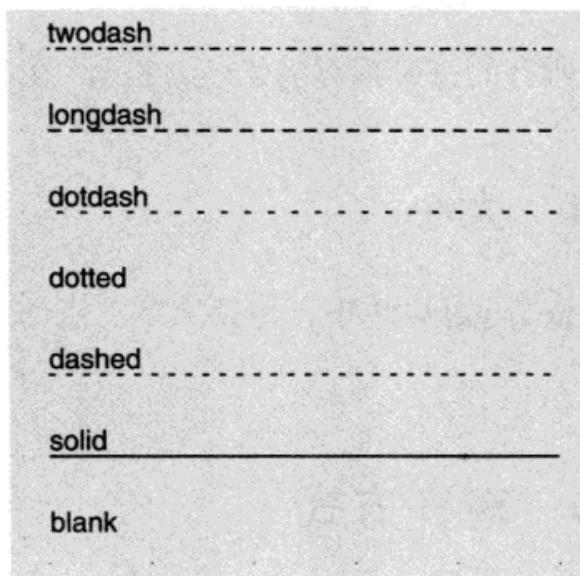
成，依次轮流表示“连续线条”和“空白”的长度。比如，字符 "33" 表示 3 个单位的连续线条 + 3 个单位的空白 (如此循环)，"3313" 表示 3 个单位的连续线条 + 3 个单位的空白，然后再接 1 个单位的连续线条 + 3 个单位的空白 (如此循环)。

上面描述的 5 种标准点划线分别对应 44, 13, 134, 73 和 2262。

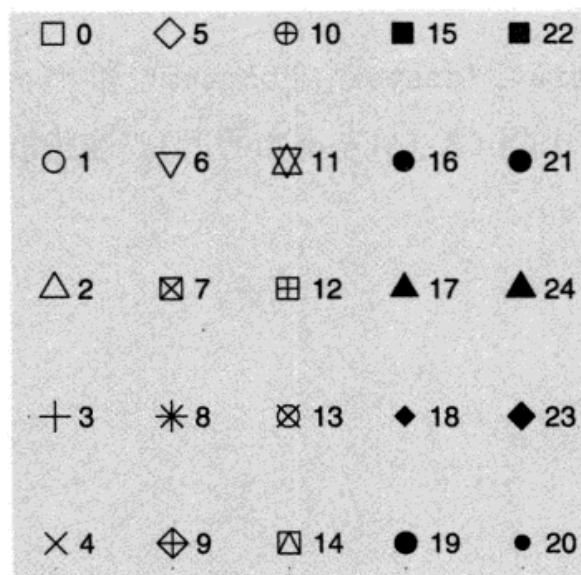
注意：NA 不是 lty 的有效值。



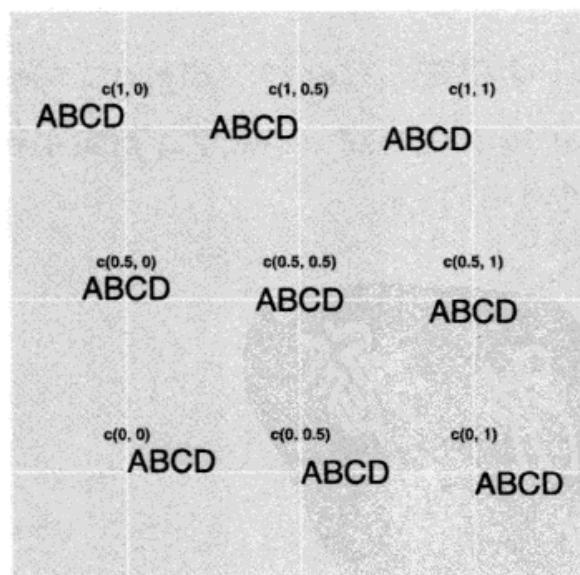
(a) LUV 色系空间中的颜色



(b) 内置的线条类型



(c) R 绘图形状。元素为黑色、背景色为蓝色。
第 25 号符号 (未画出) 可由 24 号符号翻转 180
度得到 (即黑边蓝底倒三角)



(d) 水平、竖直位置的调整

图 B.1 (参见彩图 41) 图形属性映射的展示示例。

B.3 形状

形状包括 4 种类型的值：

- $[0, 25]$ 之间的一个整数，如图 B.1(c) 所示。
- 一个单字符，用该字符作为绘图符号。
- 用一个 . 来绘制最小的可视矩形（如 1 个像素）。
- NA，不做任何绘制。

所有的符号都有前景色，符号 19 ~ 25 还具有背景色（填充）。

B.4 大小

整个 ggplot2 中，包括文本、点的大小和线条的宽度，都是以毫米为单位指定的。

B.5 对齐方式

一个字符（或图例）的对齐是在给定范围内对其位置的指定。由两种值来控制水平和竖直对齐。这些值可以是：

- 一个字符： "left", "right", "centre", "center", "bottom" 和 "top"。
- 0 和 1 之间的一个数字，指定字符串的位置（从左下角算起）。这些值如图 B.1(d) 所示。



附录 C 用 grid 操作图形

C.1 简介

有时你需要越过主题系统而直接对底层的 `grid` 图形输出进行修改。这时，你需要对 `grid` 系统有很好的理解，参见《R Graphics》一书 (Murrell, 2005)。如果你无法看到这本书，至少需阅读第 5 章：“The `grid graphics model`”，这一章是可以在网上免费获得的：<http://www.stat.auckland.ac.nz/~paul/RGraphics/chapter5.pdf>。本附录介绍 `ggplot2` 中的重要概念：视图窗口 (`viewports`) 和绘图元件 (`grobs`)，尤其是向那些需要和图形元件打交道的朋友。

C.2 视图窗口

视图窗口指定绘图的基本区域。其结构会随图形使用的分面的不同而有细微变化，但是基本区域是相同的。

面板视图窗口 (`panels viewports`) 包括图形的主要部分：条状标签、坐标轴和分面面板。视图窗口是根据它们在图形中的作用和位置来命名的。前缀描述窗口的内容，后跟整数指代 `x` 和 `y` 的位置 (从左下角算起)，中间以下划线 “`_`” 分隔开 (如下所列)。图 C.1 表示的是一个 2×2 图形的命名规则^①。

- `strip_h`: 水平条状标签
- `strip_v`: 垂直条状标签

^①译者注：`ggplot2` 0.9.0 及之后版本中命名有些出入，例如图中的 `strip_h` 和 `strip_v` 变成了 `strip_top` 和 `strip_right`，而 `axis_v` 和 `axis_h` 变成了 `axis_l` 和 `axis_b`，请载入 `grid` 包后，以 `grid.ls()` 函数看到的命名结果为基准。

- `axis_h`: 水平坐标轴
- `axis_v`: 垂直坐标轴
- `panel`: 面板

面板视图窗口包含在背景窗口 (background viewports) 中，背景窗口也包含下列窗口：

- `title`, `xlabel` 和 `ylabel`: 分别指定图形的标题, x 轴和 y 轴的标签。
- `legend_box`: 指定图形的所有图例。

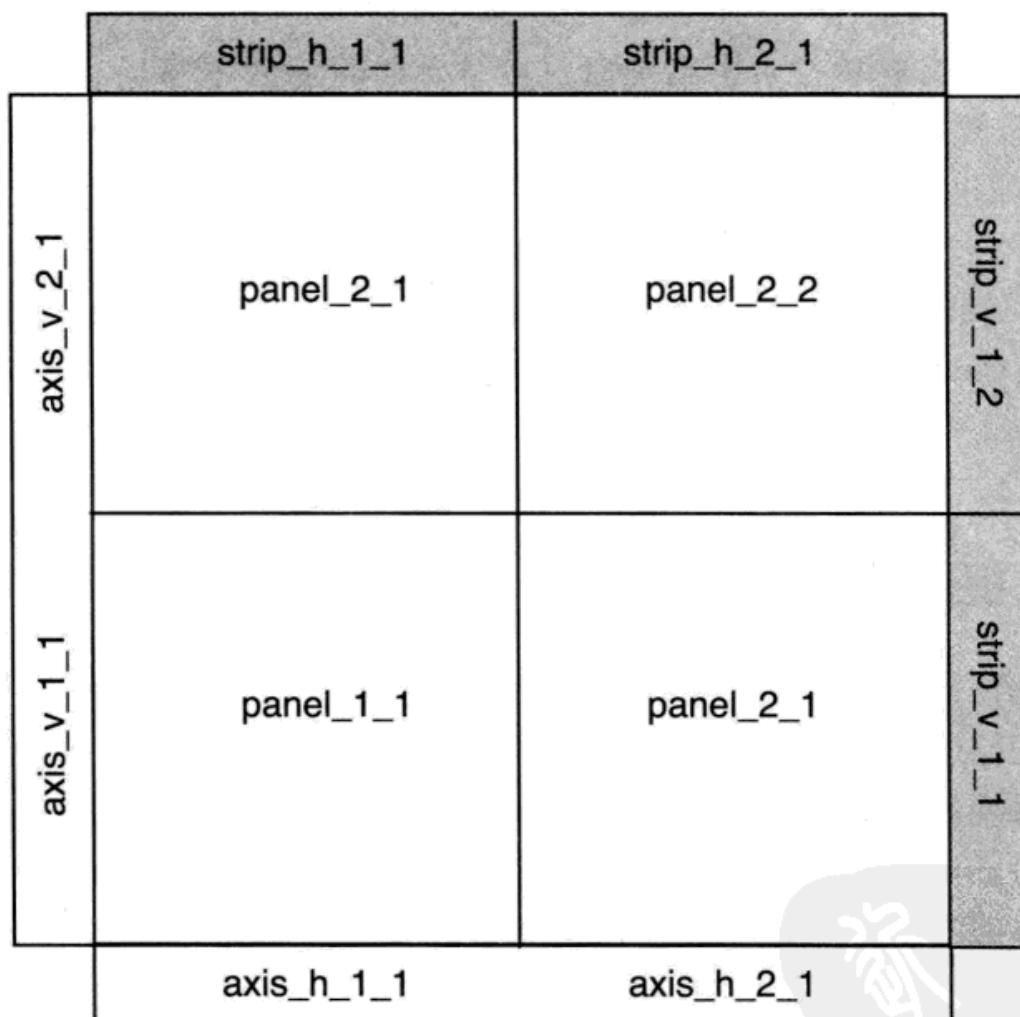


图 C.1 面板窗口视图区的命名规则。

图 C.2 中标出了一些典型的视图窗口的结构和名字。要得到当前图形的所有视图窗口列表，请运行 `current.vpTree(all=TRUE)` 或 `grid.ls(grobs = FALSE, viewports = TRUE)` (注：需加载 grid 包)。

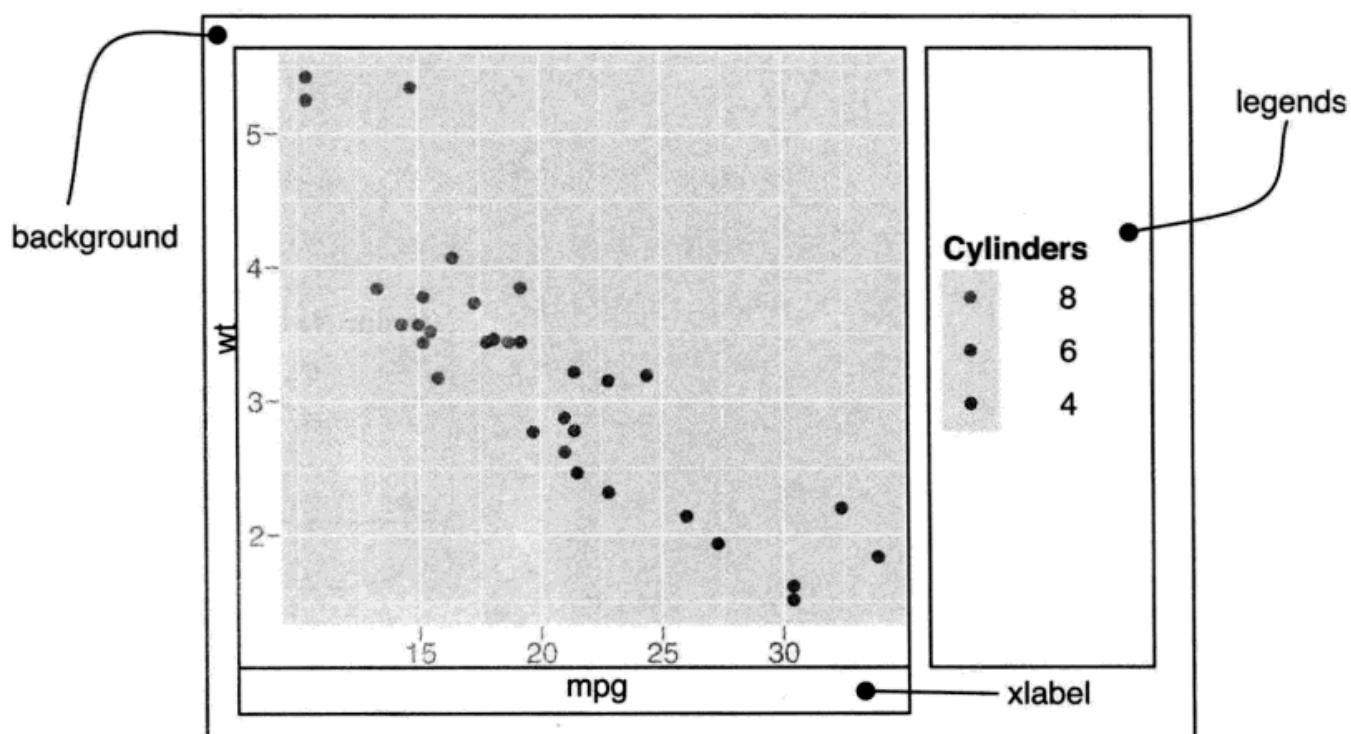


图 C.2 视图窗口的结构和名字。

C.3 绘制图形元件

图形元件名由三部分构成：元件的名字、元件的类及其唯一的数字后缀。这三个部分由“.”拼接组合在一起，比如 `title.text.435`, `ticks.segments.15`。这三个组成部分保证了所有的图形元件名都是唯一的，并且可以在同一时间用同样的名字选择多个图形元件^②。

图 C.3 标出了部分图形元件。这些元件是分层次组织在一起的，但是很难在一个示意图中把这些都表示出来。我们可以在当前窗口下输入 `grid.ls()` 命令（需加载 `grid` 包）来获取当前的图形元件列表。

^②译者注：比如可以通过正则表达式，以类似 `panel.grid*` 将匹配到的图形元件都选择上。

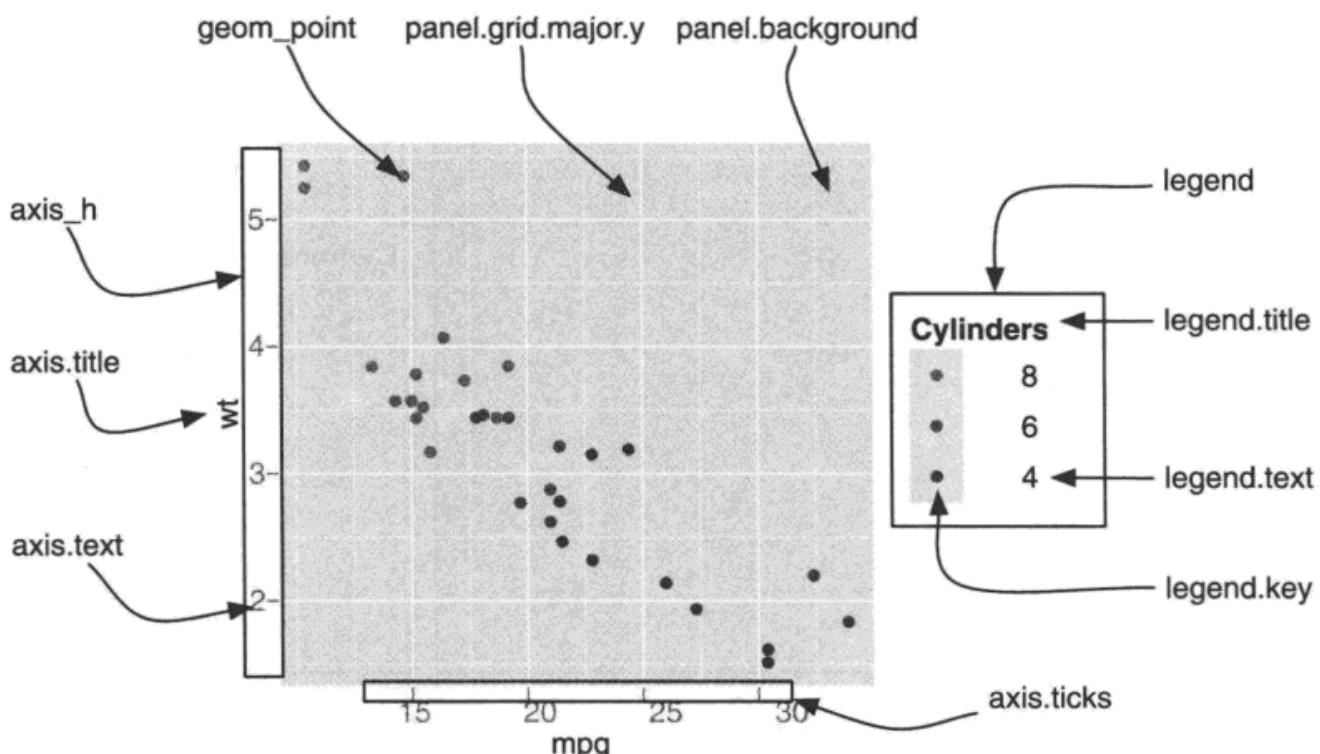


图 C.3 一些重要的图形元件。

C.4 保存工作

使用 `grid.gedit()` 以及类似功能的函数可以修改当前屏幕上的图形，但如果需要将图形存盘就需要额外的步骤；否则你会得到多张图形的输出，每张展示一处图形的变化。这里讲述的关键不是如何修改屏幕上的图形，而是修改背后的图形元件，然后根据修改后的图形元件重新绘制图形。

下面给出一个例子^③：

```

1 library(ggplot2)
2 library(grid)
3 library(grid) ## 需要 editGrob() 函数
4 p <- qplot(wt, mpg, data = mtcars, colour = cyl, main = "Title text")
5 p
6 ## 修改图形元件：图形题目字体改为斜体红色
7 g <- ggplotGrob(p)
8 idx <- which(g$layout$name == "title")
9 g$grobs[[idx]] <- editGrob(g$grobs[[idx]], gp = gpar(fontface =
10 "italic", col="red"))

```

^③译者注：此处的代码是针对 0.9.3 版本修正的。

```
11 ## 重新绘制  
12 grid.draw(g)
```

另外一种解决方案是在直接修改屏幕上的图形，然后使用 `dev.copy2pdf()` 以 pdf 格式保存最终图片。



参考文献

- A. Azzalini and A. W. Bowman. A look at some data on the old faithful geyser. *Applied Statistics*, 39:357–65, 1990.
- Cynthia A. Brewer. Guidelines for use of the perceptual dimensions of color for mapping and visualization. In *Color Hard Copy and Graphic Arts III, Proceedings of the International Society for Optical Engineering (SPIE), San Jose*, 2171:54–63, 1994a.
- Cynthia A. Brewer. *Visualization in Modern Cartography*. Color use guidelines for mapping and visualization. In A.M. MacEachren and D.R.F. Elsevier Science, 1994b.
- D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.
- Dan Carr. Using gray in plots. *ASA Statistical Computing and Graphics Newsletter*, 2:11–14, 1994. URL <http://www.galaxy.gmu.edu/~dcarr/lib/v5n2.pdf>.
- Dan Carr. *Encyclopedia of Environmetrics*, volume 2 of *Graphical displays*. John Wiley & Sons, 2002. URL <http://www.galaxy.gmu.edu/~dcarr/lib/EnvironmentalGraphics.pdf>.
- Dan Carr and Ru Sun. Using layering and perceptual grouping in statistical graphics. *ASA Statistical Computing and Graphics Newsletter*, 10(1):25–31, 1999.

- Dan Carr, Nicholas Lewin-Koh, and Martin Maechler. *hexbin: Hexagonal Binning Routines*, 2008. R package version 1.14.0.
- John M. Chambers, William S. Cleveland, Paul A. Tukey, and Beat Kleiner. *Graphical Methods for Data Analysis*. Duxbury Press, 1983. ISBN 9780534980528.
- William Cleveland. A model for studying display methods of statistical graphics. *Journal of Computational and Graphical Statistics*, 2:323–364, 1993. URL <http://stat.bell-labs.com/doc/93.4.ps>.
- William S. Cleveland. *The Elements of Graphing Data*. Wadsworth, Inc., 8 1985. ISBN 9780534037291.
- William S. Cleveland. *Visualizing Data*. Hobart Press, 1 edition, 3 1993. ISBN 9780963488404.
- William S Cleveland and Robert McGill. Graphical perception: The visual decoding of quantitative information on graphical displays of data. *Journal of the Royal Statistical Society. Series A (General)*, 150(3):192–229, 1987.
- Dianne Cook and Deborah F. Swayne. *Interactive and Dynamic Graphics for Data Analysis: With R and GGobi (Use R!)*. Springer, 1 edition, 12 2007. ISBN 9780387717616.
- John Fox. *effects: Effect Displays for Linear and Generalized Linear Models*, 2008. URL <http://socserv.socsci.mcmaster.ca/jfox/>. R package version 1.0-12.
- Jr Harrell and Frank E. *Hmisc: Harrell Miscellaneous*, 2008. URL <http://CRAN.R-project.org/package=Hmisc>. R package version 3.5-2.
- J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. URL <http://www.jstor.org/stable/2346830>.
- A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1:69–91, 1985.

Jim Lemon, Ben Bolker, Sander Oom, Eduardo Klein, Barry Rowlingson, Hadley Wickham, Anupam Tyagi, Olivier Eterradossi, Gabor Grothendieck, Michael Toews, and John Kane. *plotrix: Various plotting functions*, 2008. URL <http://CRAN.R-project.org/package=plotrix>. R package version 2.4-3.

Thomas Lumley. *dichromat: Color schemes for dichromats*, 2007. R package version 1.2-2.

Doug McIlroy. *mapproj: Map Projections*, 2005. R package version 1.1-7.1. Packaged for R by Ray Brownrigg and Thomas P Minka.

David Meyer, Achim Zeileis, and Kurt Hornik. The strucplot framework: Visualizing multi-way contingency tables with vcd. *Journal of Statistical Software*, 17(3):1–48, 10 2006. URL <http://www.jstatsoft.org/v17/i03>.

Paul Murrell. *Investigations in graphical statistics*. PhD thesis. The University of Auckland, 1998. URL <http://books.google.com.hk/books?id=v4-UMgAACAAJ>.

Paul Murrell. *R Graphics*. Chapman & Hall/CRC, 2005.

Naomi B. Robbins. *Creating More Effective Graphs*. Wiley-Interscience, 1 edition, 12 2004. ISBN 9780471274025.

Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008a. URL <http://lmdvr.r-forge.r-project.org>. ISBN 978-0-387-75968-5.

Deepayan Sarkar. *lattice: Lattice Graphics*, 2008b. URL <http://CRAN.R-project.org/package=lattice>. R package version 0.17-6.

Edward R. Tufte. *Envisioning Information*. Graphics Pr, 5 1990. ISBN 9780961392116.

Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, 2 1997. ISBN 9780961392123.

- Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition, 5 2001. ISBN 9780961392147.
- Edward R. Tufte. *Beautiful Evidence*. Graphics Pr, 1st edition edition, 7 2006. ISBN 9780961392178.
- John W. Tukey. *Exploratory Data Analysis*. Addison Wesley, 1 edition, 1977. ISBN 9780201076165.
- Gregory R. Warnes. *gplots: Various R programming tools for plotting data*, 2007. URL <http://CRAN.R-project.org/package=gplots>. R package version 2.6.0. Includes R source code and/or documentation contributed by Ben Bolker and Thomas Lumley.
- Edward J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):pp. 664–675, 1990. URL <http://www.jstor.org/stable/2290001>.
- Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL <http://www.jstatsoft.org/v21/i12/>.
- Hadley Wickham. *Practical tools for exploring data and models*. PhD thesis, PhD thesis, Iowa State University, 2008. URL <http://had.co.nz/thesis>.
- Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.
- Hadley Wickham, Michael Lawrence, Duncan Temple Lang, and Deborah F Swayne. An introduction to rggobi. *R-news*, 8(2):3–7, October 2008. URL http://CRAN.R-project.org/doc/Rnews/Rnews_2008-2.pdf.
- Leland Wilkinson. *The Grammar of Graphics*. Statistics and Computing. Springer, 2nd edition, 7 2005. ISBN 9780387245447.
- Achim Zeileis, Kurt Hornik, and Paul Murrell. Escaping rgbland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, 53(9):3259–3270, 2009. Forthcoming.

主题索引

- 3d graphics (三维图像), 82
- Adjustment (调整)
 - position (位置), 62
- Aesthetics(图形属性), 11
 - defaults (默认值), 156
 - mappings (映射), 参见 Mappings
 - matching to geoms (匹配几何对象), 54
 - setting (设置), 12, 50, 54
 - specifications (定义), 200
 - translating from qplot (从qplot转换), 190
- Annotation (注解), 91
- Appendices (附录), 190
- Area plot (面积图), 70
- Aspect ratio (纵横比), 145
- Axis (坐标轴)
 - breaks (断点), 104
 - expansion (扩展), 106
 - formatting (格式化), 105
 - labels (标签), 25, 104
 - limits (范围), 25, 104, 142
 - multiple (多个), 173
- Background (背景), 152
- Barchart (条形图), 70
- Base graphics (基础图形), 4
 - translating from (从... 转换), 192
- Bibliography (参考文献), 208
- Binning (封箱)
 - 1d (一维), 参见 Histogram (直方图)
 - 2d (二维), 80
- Boxplot (箱线图), 13, 73
- Bugs (漏洞), 7
- Clustering (聚类), 176
- Colour (颜色), 110
- blindness (色盲), 111
- Brewer (调制), 114
- discrete scale (离散标度), 114
- gradients (渐变), 111
- palettes (调色板), 115
- spaces (色彩空间), 111
- specifying (设定), 200
- transparency (透明度), 77
- wheel (色轮), 33
- Conditional density plot(条件密度图), 72
- Converting data (变换数据)
 - from wide to long (从宽变长), 171
- Coordinate systems (坐标系), 139
 - Cartesian (笛卡尔系), 141
 - Equal scales (相同标度), 144
 - flipping the axes (翻转坐标), 142
 - interaction with scales(与标度结合), 142
 - introduction (导论), 39
 - map projections (地图投影), 146
 - non-Cartesian (非笛卡尔系), 145
 - polar (极坐标), 145
 - transformation (变换), 144
- Data (数据), 46
 - diamonds, 9
 - mpg, 29
 - msleep, 99
 - longitudinal (纵向), 23, 52
 - replacing (替换), 47
 - wide-to-long (宽变长), 171
- Date (日期), 108
- Density (密度)
 - 2d (二维), 80
 - plot (图形), 18, 77
- Distributions (分布), 72

Dodging (并列), 62
vs. faceting (vs 分面), 133

Duplication (重复)
functions (函数), 188
iteration (迭代), 184
reducing (减少), 184
templates (模板), 184

Error band (误差带), 85

Error bar (误差棒), 85

Exporting (导出), 157
to Latex (到 LATEX), 159
to Powerpoint (到 Powerpoint), 159
to Word (到 Word), 159

Faceting (分面), 23, 123
by continuous variables (连续变量), 136
controlling scales (标度控制), 130
grid (网格型), 124
interaction with scales (和标度结合), 130
margins (边际), 127
missing data (缺失数据), 133
panel size (面板大小), 131
vs. dodging (vs 并列), 135
vs. grouping (vs 分组), 133
wrapped (封装型), 129

Fortify (增补), 178

Frequency polygon (频率多边形), 72

Functions that create plots (绘图函数), 188

Geoms (几何对象)

aliases (别名), 64
combining with stats (和统计变换结合), 63
customising defaults (自定义默认值), 156
defaults (默认值), 57
parameterisation (参数化), 57, 140

ggplot

data structures (数据结构), 40
methods (方法), 178
translating from qplot (从 qplot 转换), 190

GPL

translating from (从... 转换), 198

Grammar (语法)

components (组件), 37
introduction (介绍), 3

Grid (网格), 5, 121, 200
grobs (绘图元件), 203
viewports (视图窗口), 203
Grouping (分组), 51
vs. faceting (vs 分面), 133
Guides (引导元素), 98

Histogram (直方图), 14
choosing bins (选择组距), 72
weighted (权重), 95

Image plot (色深图), 71
Installation (安装), 7
Interactive graphics (交互图形), 4
Iteration (迭代), 184

Jittering (扰动), 17, 75, 78

Labels (标签), 94
Lattice graphics (Lattice 图形), 5
translating from (从... 转换), 196
Layers (图层)
adding (添加), 43
components (组件), 38
reusing (重复使用), 186
saving as variables (存储为变量), 46
strategy (策略), 69

Layout (布局), 159

Legend (图例), 119
keys (符号), 121
position (位置), 121
title (标题), 121

Level plot (水平图), 71

Limits (范围), 参见 Axis, limits

Line plot (线条图), 70

Line type (线条类型)
scale (标度), 115
specifying (设置), 201

Linear models (线性模型), 179

Log transform (对数变换), 25, 107

Longitudinal data (纵向数据), 参见 Time series

Mailing list (邮件列表), 7

Mappings (映射), 30, 47
creating programmatically(语法上创建), 188
defaults (默认), 48

- overriding (重置), 48
- Maps (地图)
 - borders (边界), 82
 - choropleth (等值线), 83
 - drawing (绘制), 82
 - projections (投影), 146
- Model (模型)
 - diagnostics (诊断), 66, 179
 - fitting multiple models (拟合多个模型), 168
 - generalised additive (广义可加), 15
 - linear (线性), 15, 179
 - loess (局部多项式回归), 15
 - robust (稳健), 16
- Munching (分割再组合), 140
- Named plots (图形名称), 32
- Overplotting (遮盖绘制), 77
- Package (包)
 - effects, 85
 - grid, 203
 - MASS, 17
 - mgcv, 15, 169
 - multcomp, 87
 - multcompView, 87
 - nlme, 51
 - plyr, 164
 - reshape2, 171
- Parallel coordinates plot (平行坐标图), 175, 189
- Polar coordinates (极坐标图), 145
- Position adjustments (位置调整), 62
- Positioning(位置), 123
 - coordinate systems (坐标系), 139
 - faceting (分面), 123
 - position adjustments (位置调整), 62
 - scales (标度), 105
- Publication (出版)
 - multiple plots on the same page (一页多图), 159
 - polishing plots for (精雕细琢图片), 147
 - saving output (保存输出), 157
 - themes (主题), 147
- qplot
 - getting started (开始), 10
- other options (其他选项), 25
 - 与plot()的区别, 26
 - 转换为ggplot(), 190
- Reducing duplication (减少重复), 184
- Rescaling (标度重置), 175
- Rotating (旋转), 142
- Saving (保存), 157
- grid graphics (grid 图形), 206
- Scales (标度)
 - adding (添加), 100
 - colour (颜色), 110
 - continuous (连续型), 111
 - discrete (离散型), 114
 - gradient (渐变), 111
 - customising defaults (自定义默认设置), 156
 - date-time (日期 -时间型), 108
 - defaults (默认), 100
 - identity (同一型), 119
 - interaction with coordinatesystem (和坐标系结合), 142
 - interaction with facetting (和分面结合), 130
 - introduction (介绍), 38
 - limits (范围), 104
 - mapping (映射), 100
 - names (名称), 103
 - position (位置)
 - continuous (连续型), 107
 - discrete (离散型), 110
 - training (训练), 100
 - transformation (变换), 99
- Scatterplot (散点图), 10
 - principles of (原则), 30
- Shape (形状)
 - scale (标度), 115
 - specifying (设置), 202
- Side-by-side (并列), 参见 Dodging
- Size (大小)
 - scale (标度), 115
 - specifying (设定), 202
- Smoothing (平滑), 14
- Stacking (堆叠), 62
- Stats (统计变换)
 - combining with geoms (和几何对象结合), 63

- creating new variables (创建新变量), 60
precomputed (预先计算), 65
summary (摘要), 89
Sub-figures (子图)), 160
Subplots (子图)), 160
Summary (摘要)
 statistical(统计), 89
Summary (汇总)
 group-wise (分组), 166
 of plot object (图形对象), 45
Surface plots (曲面图), 82
- Templates (模板), 185
Themes (主题), 147
 background (背景), 152
 built-in (内置), 148
 elements (元素), 150
 labels (标签), 151
 updating (更新), 154
 white background (白色背景), 148
- Time (时间), 108
Time series (时间序列)
 bivariate (双变量), 21
 date formatting (数据格式), 109
 multivariate (多重), 172
- Title (标题), 25
Transformation (变换)
 column-wise (列), 166
 coordinate system (坐标系统), 139, 144
 data (数据), 108
 group-wise (组), 166
 scales (标度), 108
- Translating (转换)
 from base graphics (从基础图形), 192
 from GPL (从 GPL), 198
 from lattice (从 lattice), 196
 from qplot (从 qplot), 190
- Transparency (透明度), 77
Uncertainty (不确定性)
 visualising (可视化), 85
- Viewports (视图窗口), 203
- Website (网站), 2
Weighting (权重), 95

函数索引

+, 43, 48, 101
%+%, 47
.., 61

aes(), 47
aes_string(), 188
alpha(), 78

catcolwise(), 166
colwise(), 166
coord_cartesian(), 139
coord_equal(), 144
coord_flip(), 143
coord_map(), 146
coord_polar(), 145
coord_trans(), 144
cut_interval, 136
cut_number(), 136

data.frame, 46
ddply(), 164

facet_grid(), 124
facet_wrap(), 124, 129
fortify(), 178
fortify.Image(), 182
fortify.lm(), 179

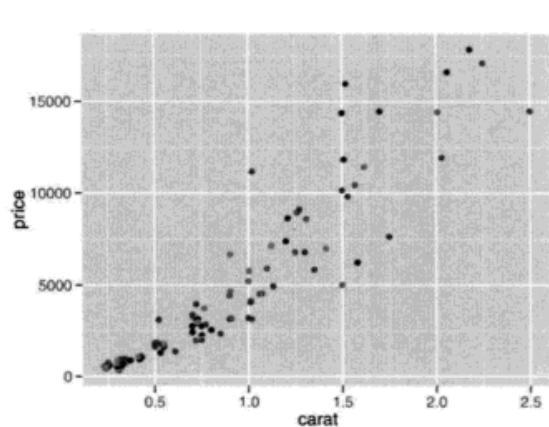
geom_abline(), 94
geom_area(), 70, 71, 75
geom_bar(), 20, 70
geom_boxplot(), 17, 54, 75
geom_crossbar(), 86
geom_density(), 18, 76
geom_density2d(), 81
geom_errorbar(), 86
geom_freqpoly(), 73
geom_ggsave(), 158
geom_hexagon(), 80
geom_histogram(), 18, 73
geom_hline(), 94
geom_jitter(), 18, 75, 79
geom_line(), 54, 55, 65, 71, 94, 173, 175
geom_linerange(), 86
geom_path(), 21, 55, 71
geom_point(), 71, 77–79, 81, 83
geom_pointrange(), 86
geom_polygon(), 71, 85
geom_rect(), 94
geom_ribbon(), 86
geom_save(), 41
geom_smooth(), 14, 45, 46, 53, 66, 67, 86
geom_text(), 71
geom_tile(), 71
geom_vline(), 94
ggplot(), 43
ggsave(), 159
I(), 12, 18
last_plot(), 184

layer(), 43
load(), 47
melt(), 171
numcolwise(), 166
plot(), 8
position_dodge(), 62
position_fill(), 62
position_jitter(), 62, 79
position_stack(), 62
print(), 40
qplot(), 10
save(), 40
scale_colour_brewer(), 115
scale_colour_gradient(), 105, 111
scale_colour_gradient2(), 112
scale_colour_gradientn(), 113
scale_colour_hue(), 114
scale_colour_manual(), 115
scale_fill_gradient(), 111
scale_fill_gradient2(), 112
scale_fill_gradientn(), 113
scale_identity(), 119
scale_linetype(), 115
scale_linetype_manual(), 115
scale_shape(), 115
scale_shape_manual(), 115
scale_size_discrete(), 115
scale_x_continus(), 107
scale_x_log10(), 108
scale_y_continus(), 107
scale_y_log10(), 108
stat_bin(), 73
stat_bin2d(), 80
stat_boxplot(), 73
stat_density(), 65, 75
stat_density2d(), 81
stat_hexbin(), 80
stat_smooth(), 169
stat_summary(), 89
subset(), 165
summary(), 40
element_blank(), 153
element_line(), 152
element_text(), 151
theme(), 154
theme_bw(), 149
theme_get(), 154
theme_grey(), 149
theme_set(), 149
transform(), 165
update_geom_defaults(), 156
update_stat_defaults(), 156
xlim(), 105
ylim(), 105

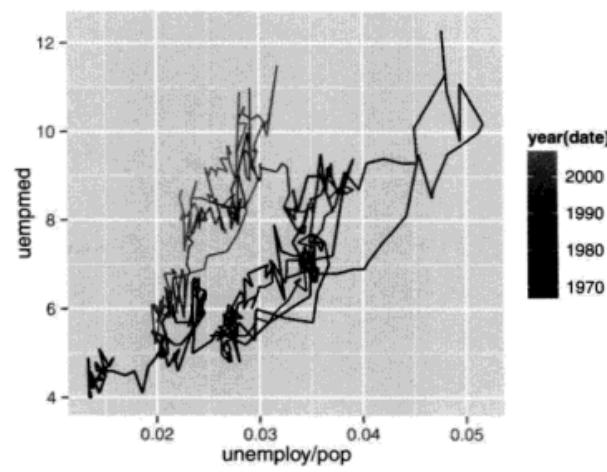


彩色插图

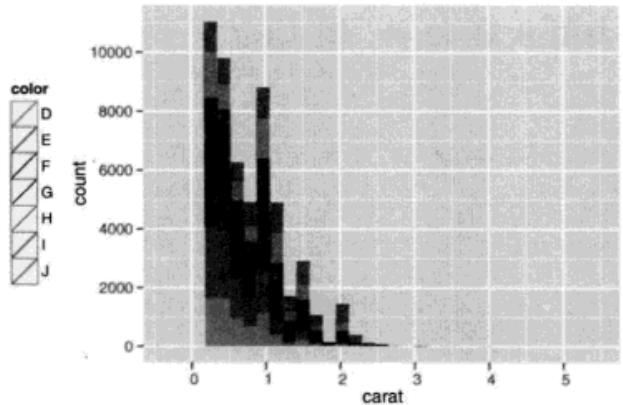
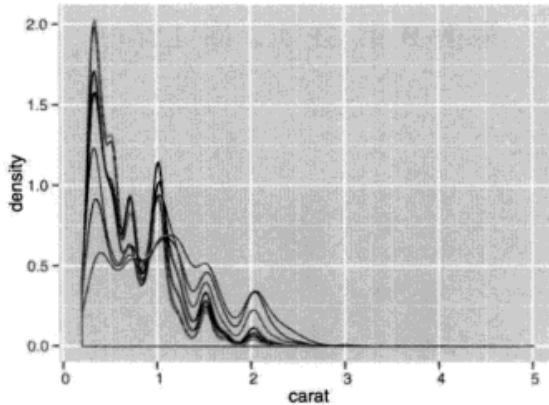




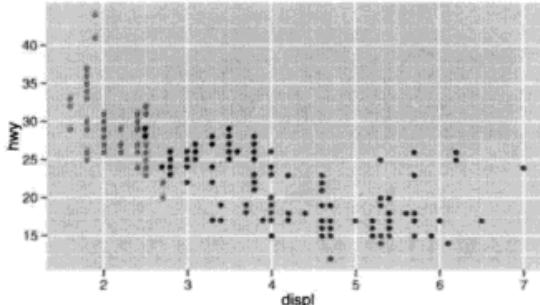
彩图 1 第 12 页, 图 2.2(左)



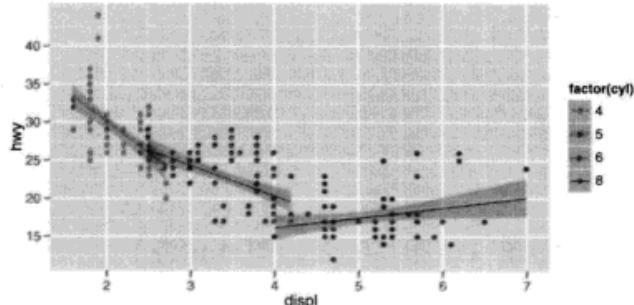
彩图 2 第 22 页, 图 2.15(右)



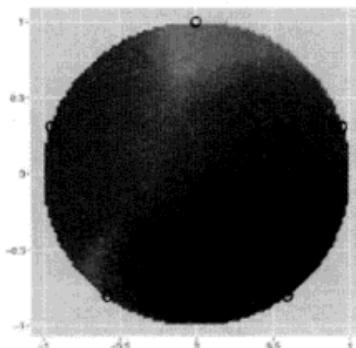
彩图 3 第 20 页, 图 2.12



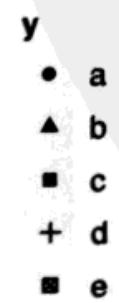
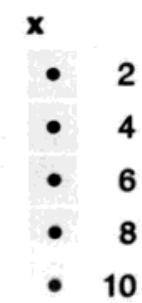
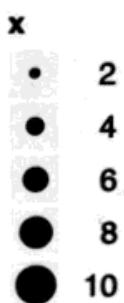
彩图 4 第 30 页, 图 3.1



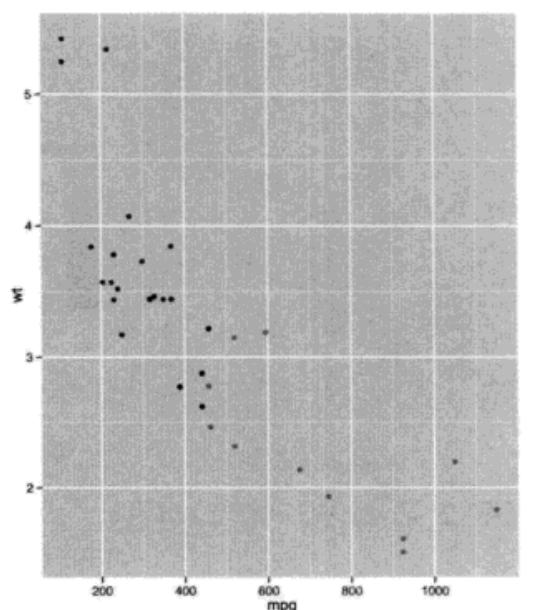
彩图 5 第 32 页, 图 3.3



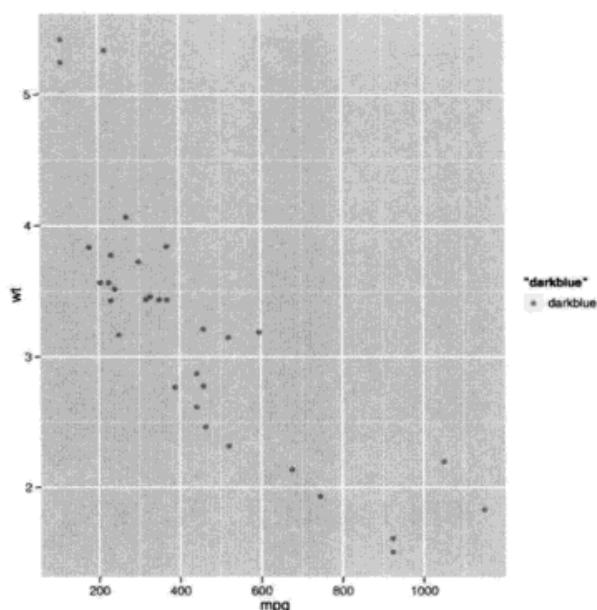
彩图 6 第 33 页, 图 3.4



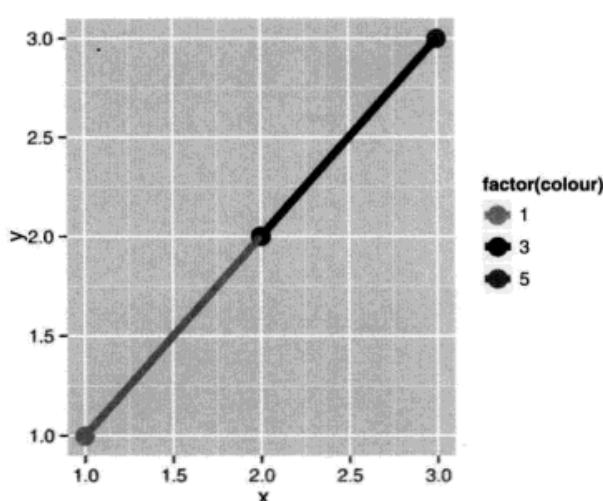
彩图 7 第 39 页, 图 3.8



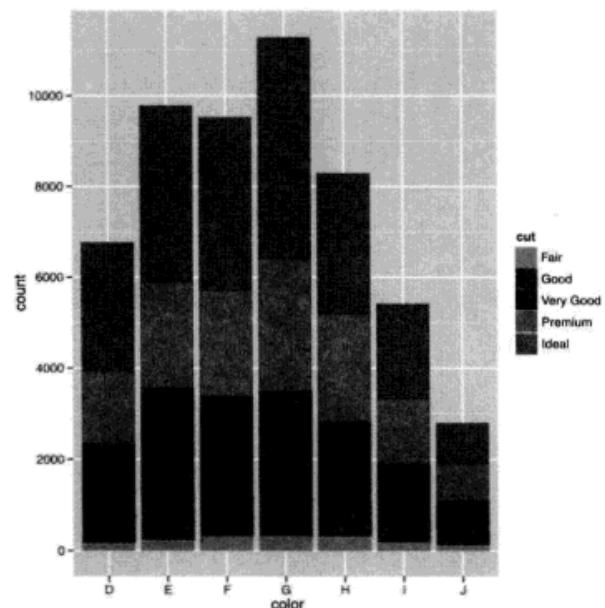
彩图 8 第 49 页, 图 4.1(左)



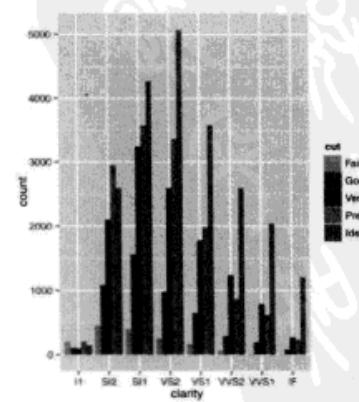
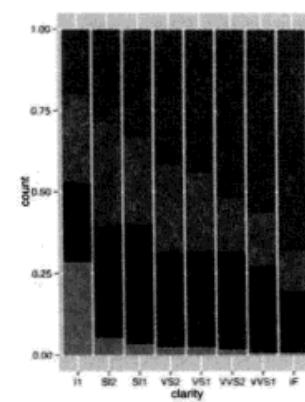
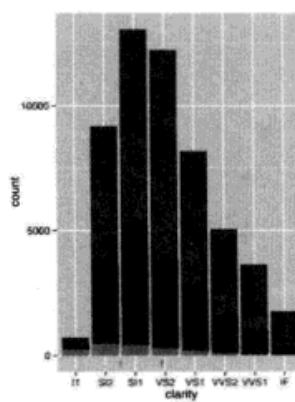
彩图 9 第 51 页, 图 4.2(右)



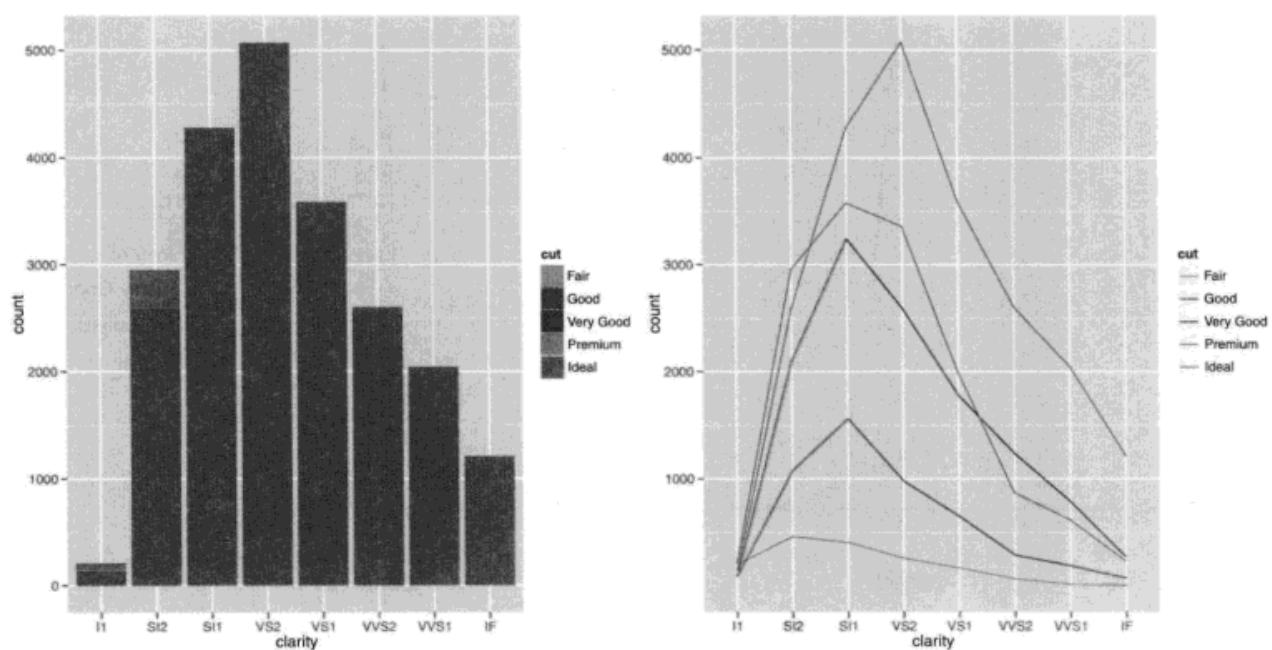
彩图 10 第 55 页, 图 4.6(左)



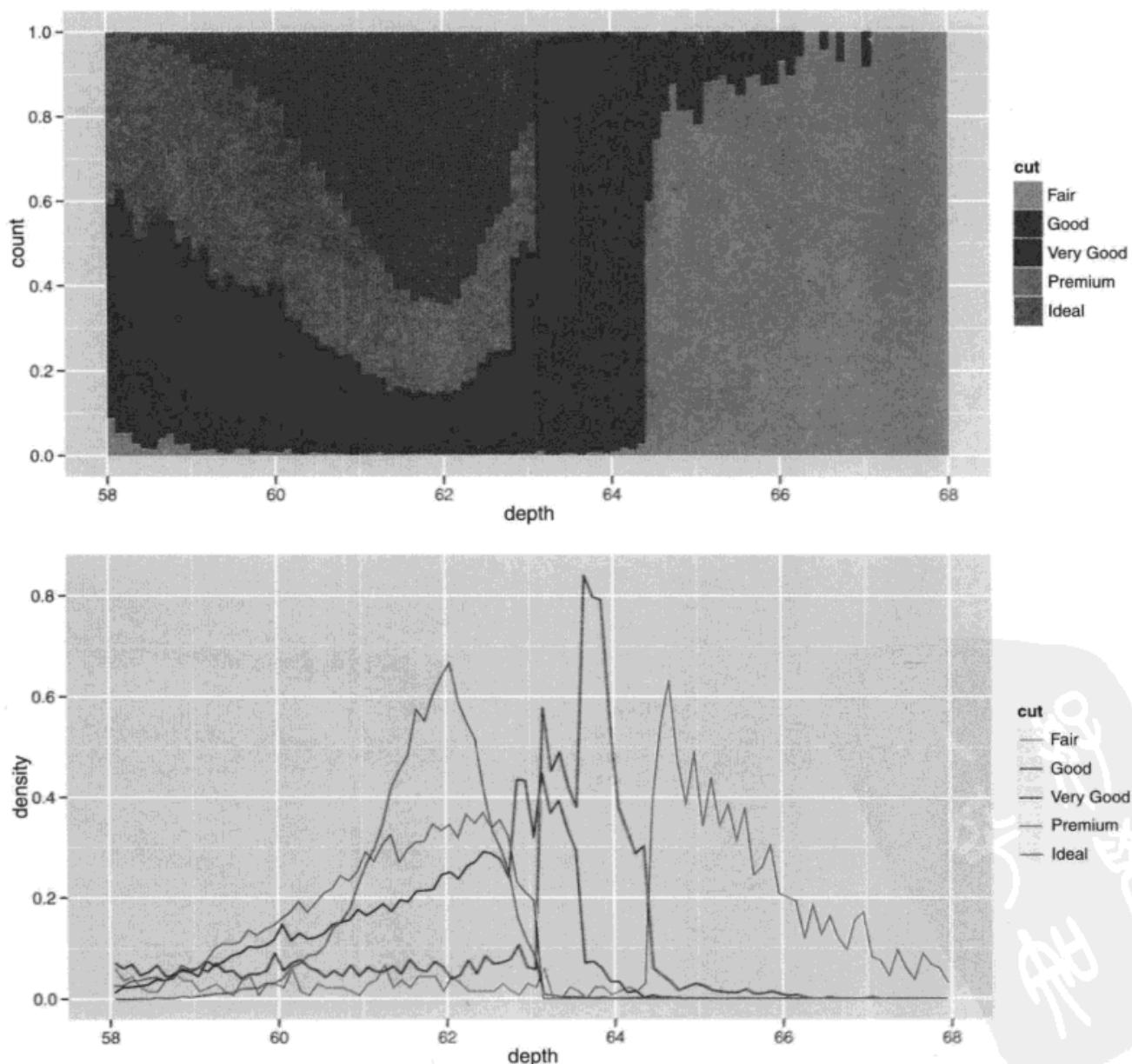
彩图 11 第 56 页, 图 4.7(右)



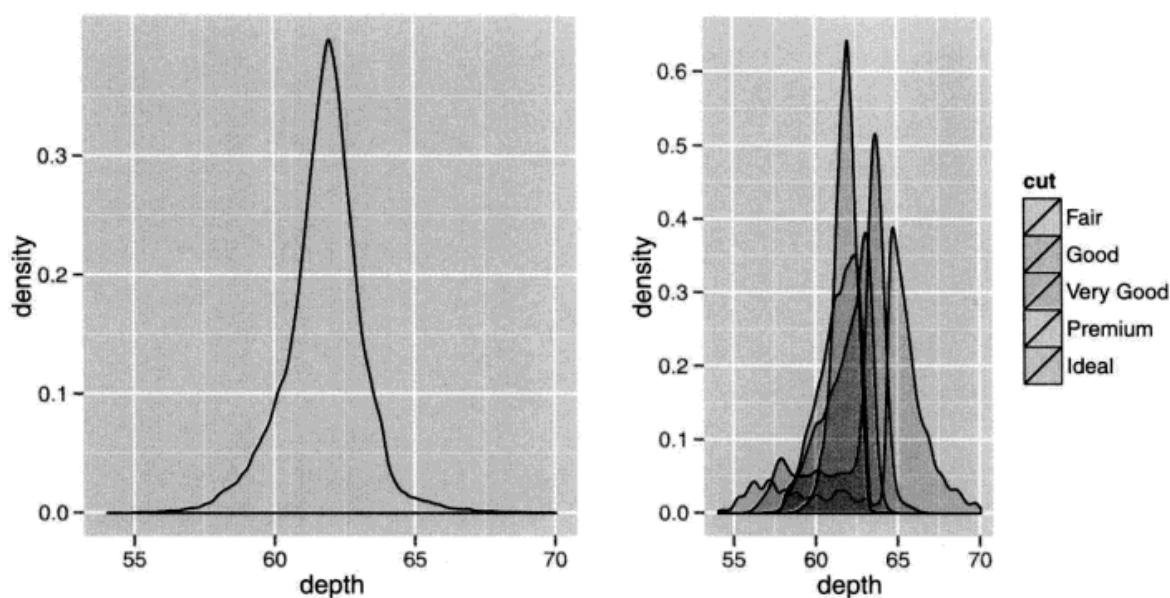
彩图 12 第 62 页, 图 4.8



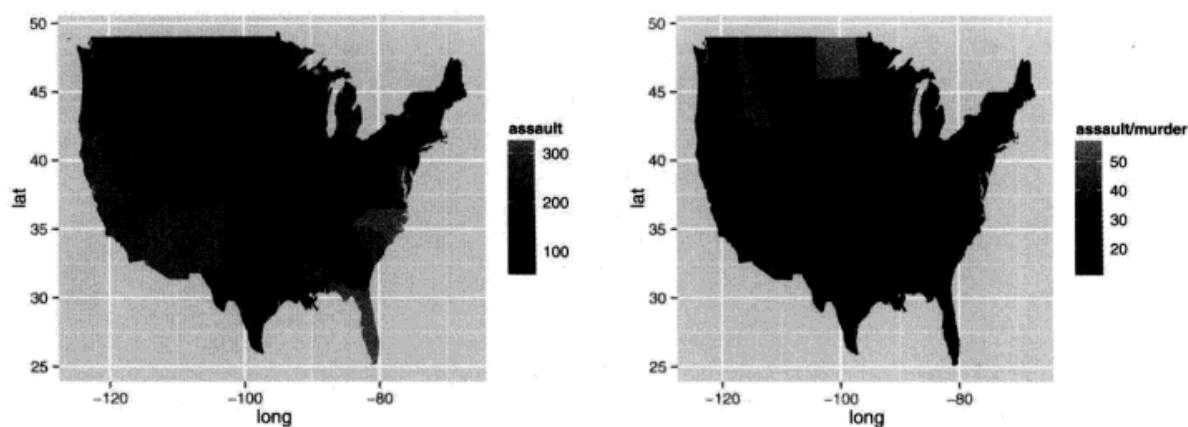
彩图 13 第 63 页, 图 4.9



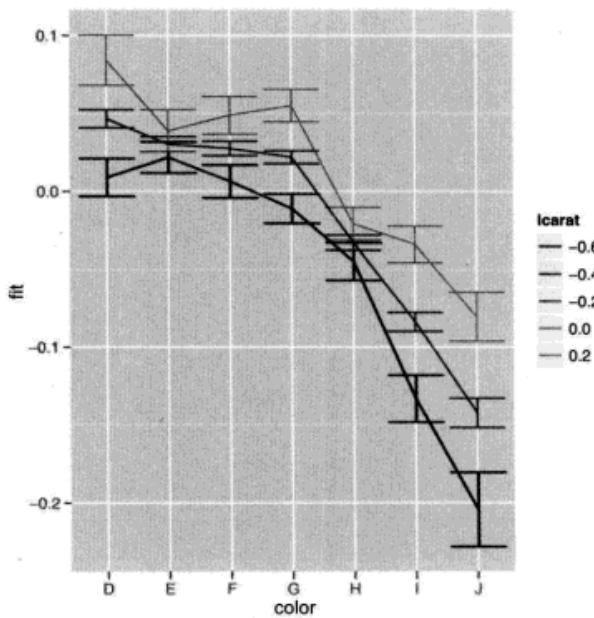
彩图 14 第 74 页, 图 5.3(中、下)



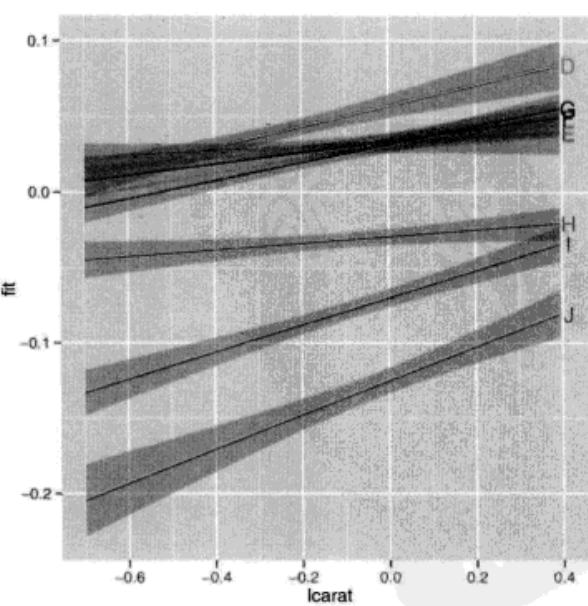
彩图 15 第 77 页, 图 5.6



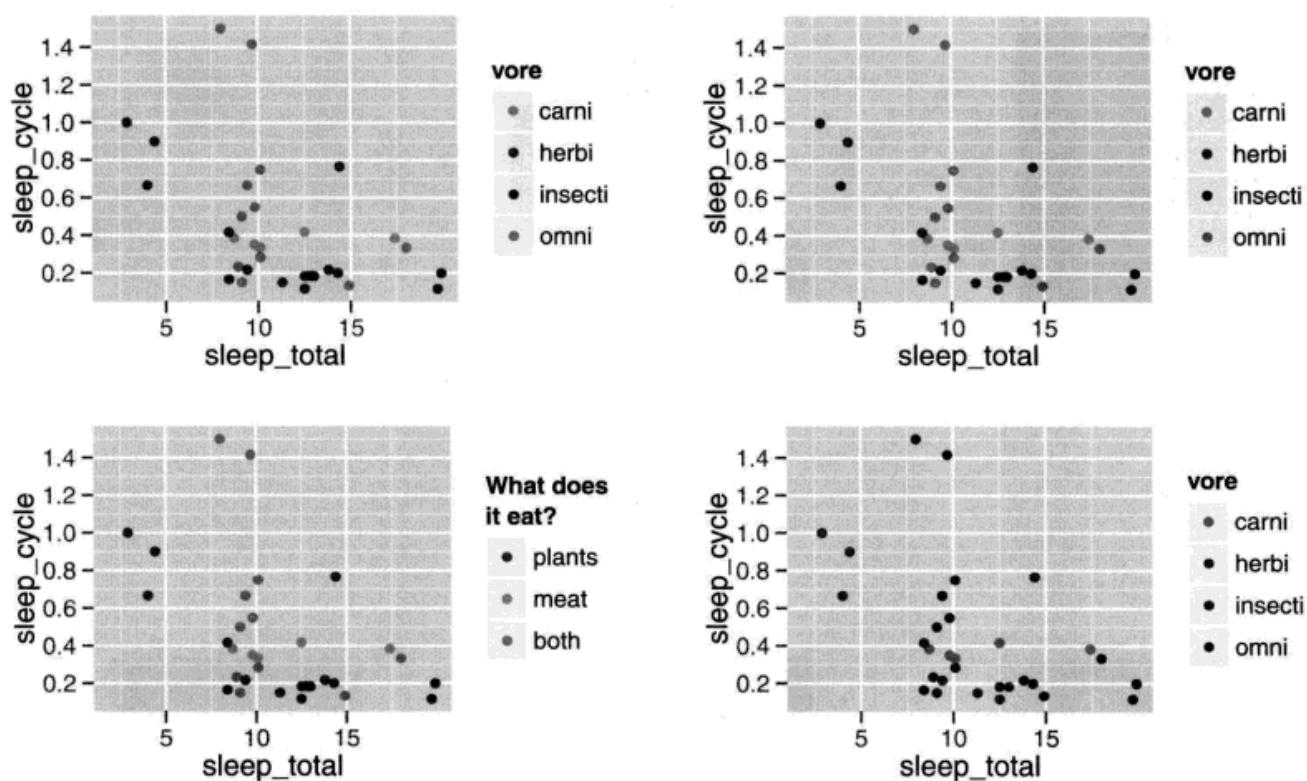
彩图 16 第 84 页, 图 5.13



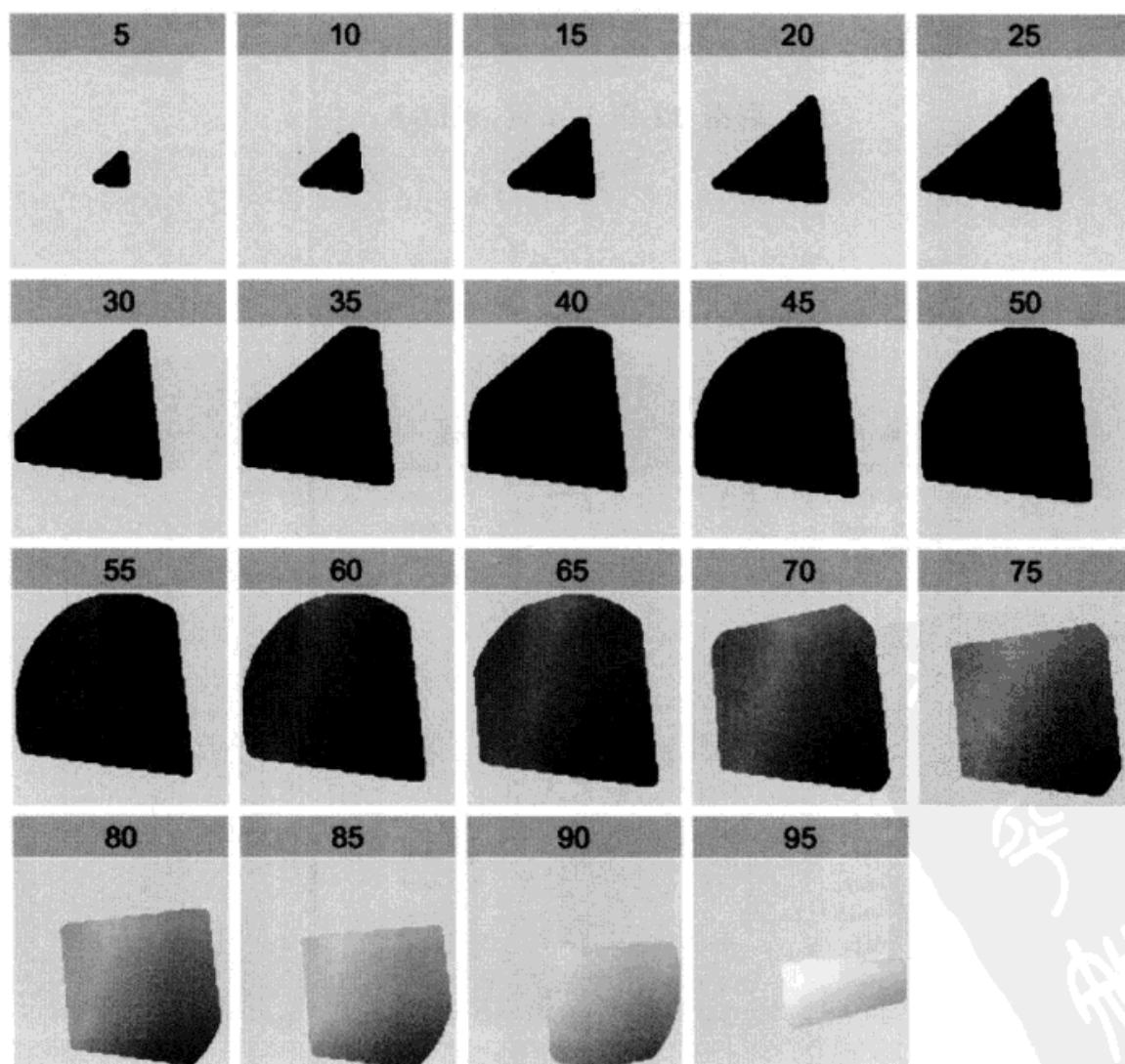
彩图 17 第 88 页, 图 5.15(右)



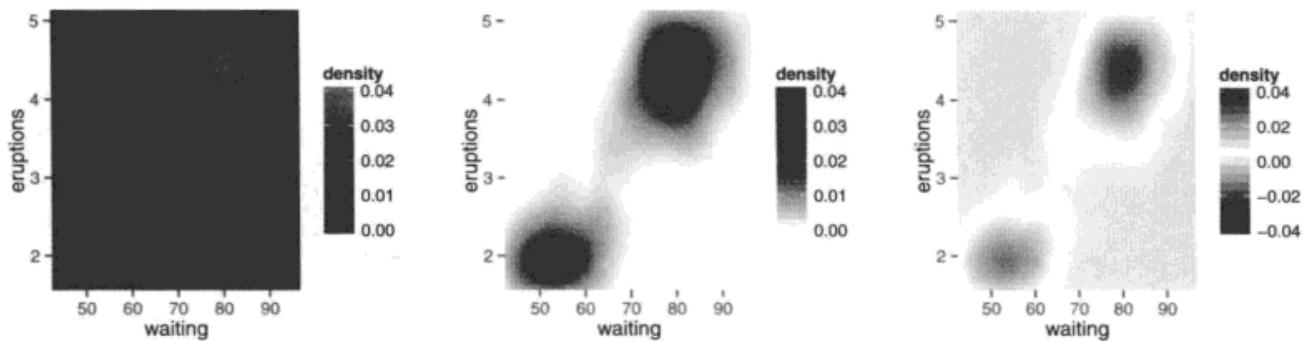
彩图 18 第 88 页, 图 5.16(右)



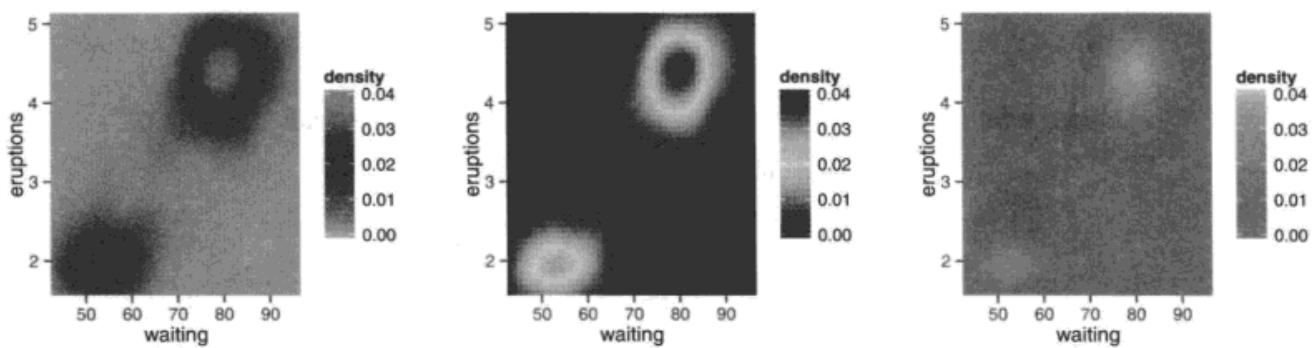
彩图 19 第 102 页, 图 6.1



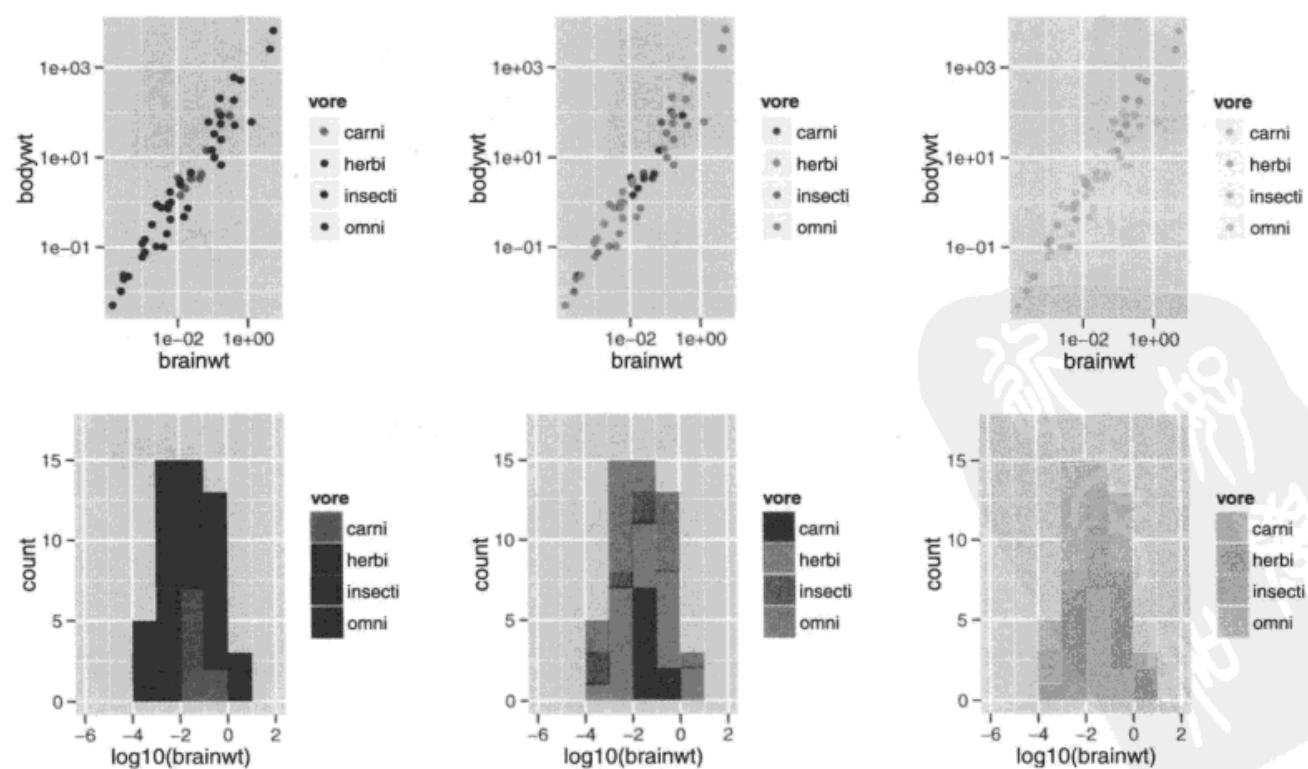
彩图 20 第 112 页, 图 6.6



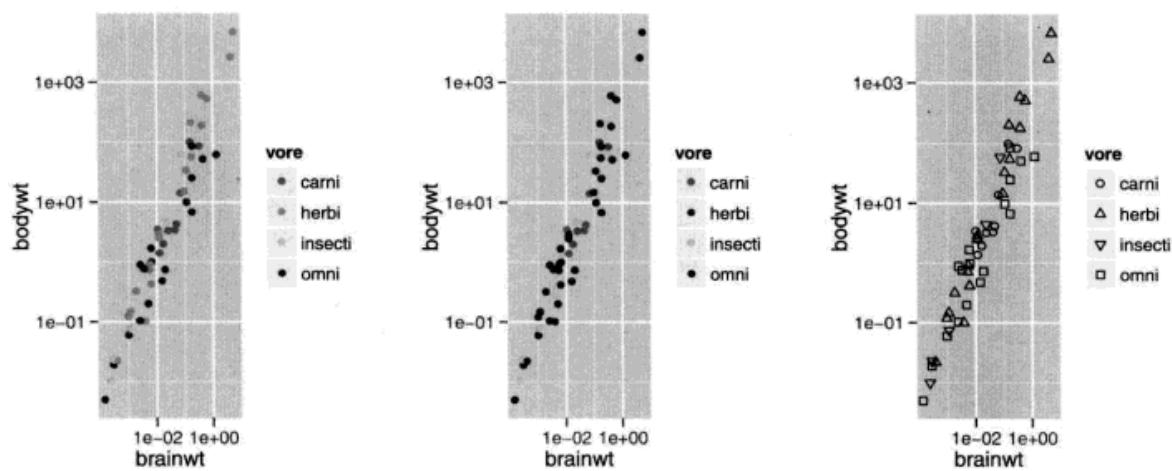
彩图 21 第 113 页, 图 6.7



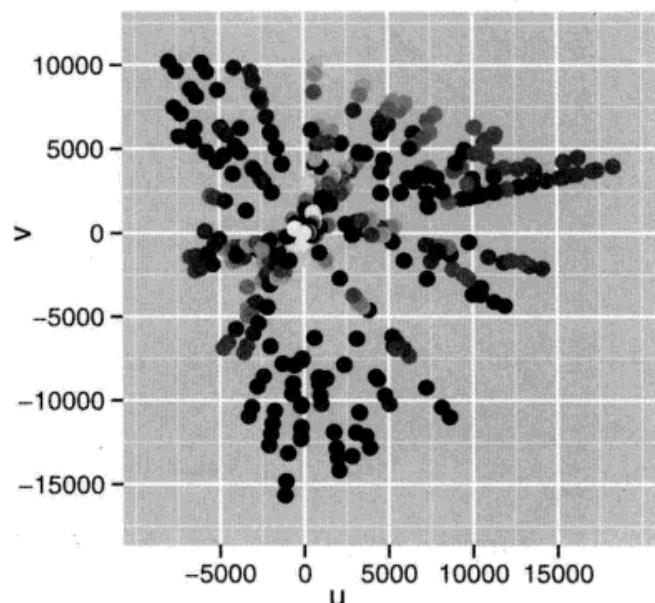
彩图 22 第 114 页, 图 6.8



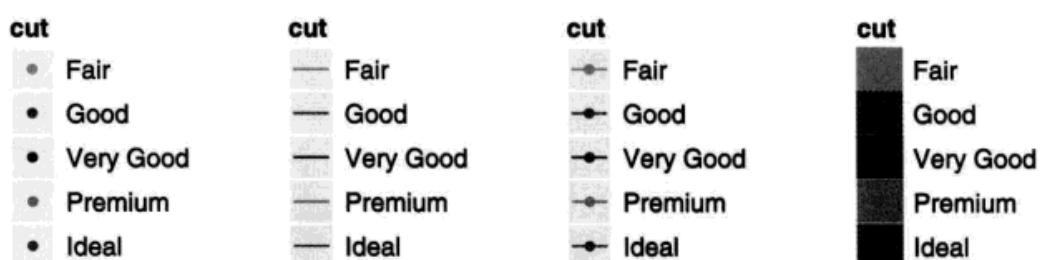
彩图 23 第 116 页, 图 6.9



彩图 24 第 116 页, 图 6.10



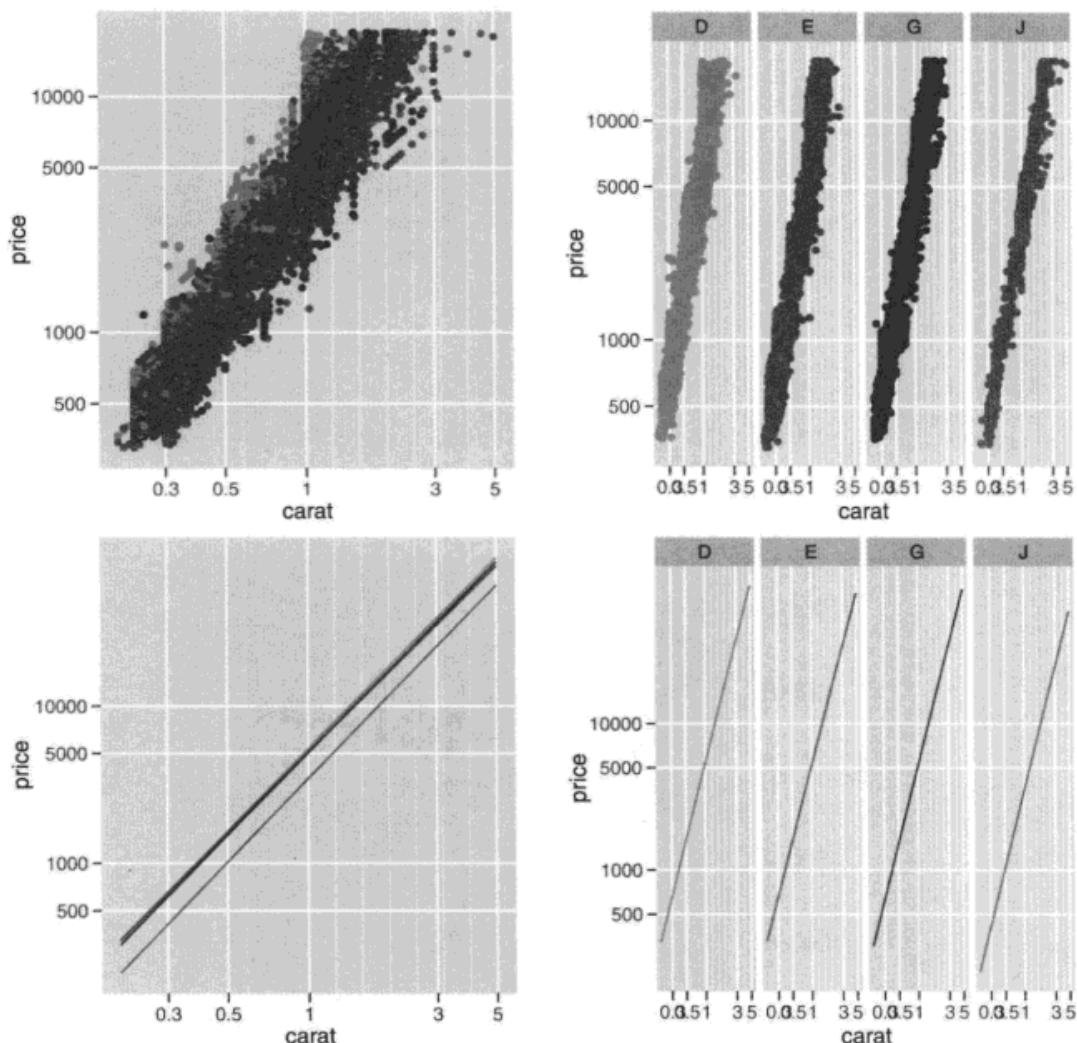
彩图 25 第 119 页, 图 6.11



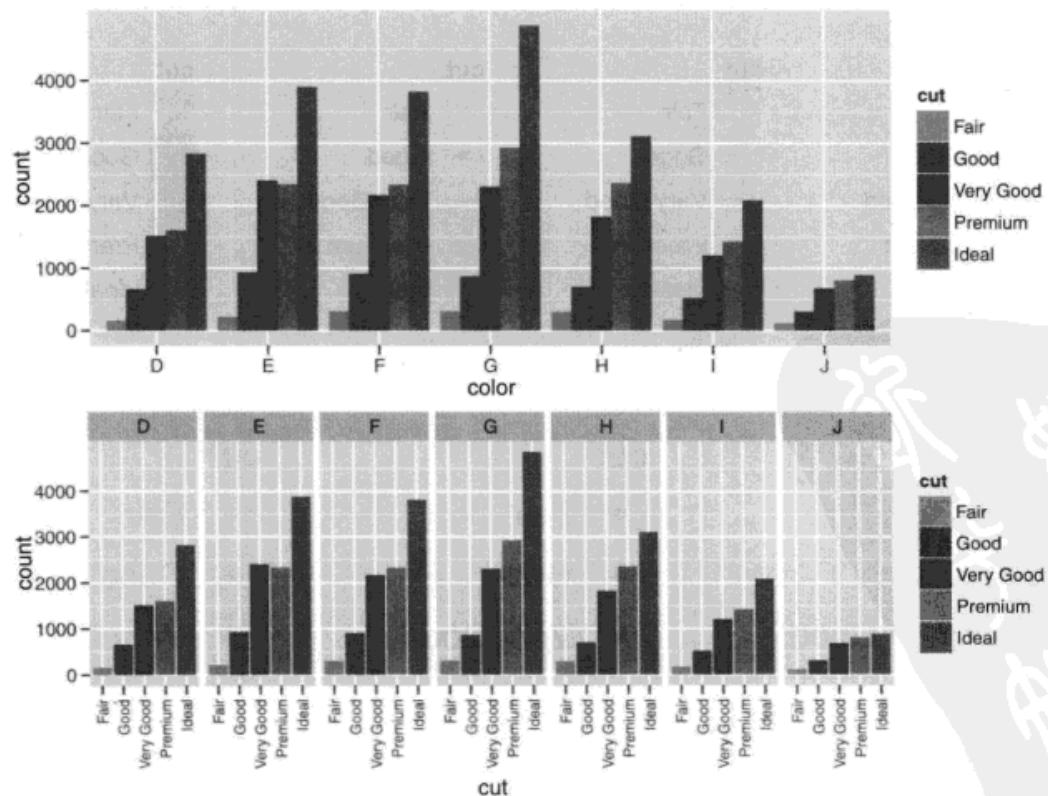
彩图 26 第 120 页, 图 6.13



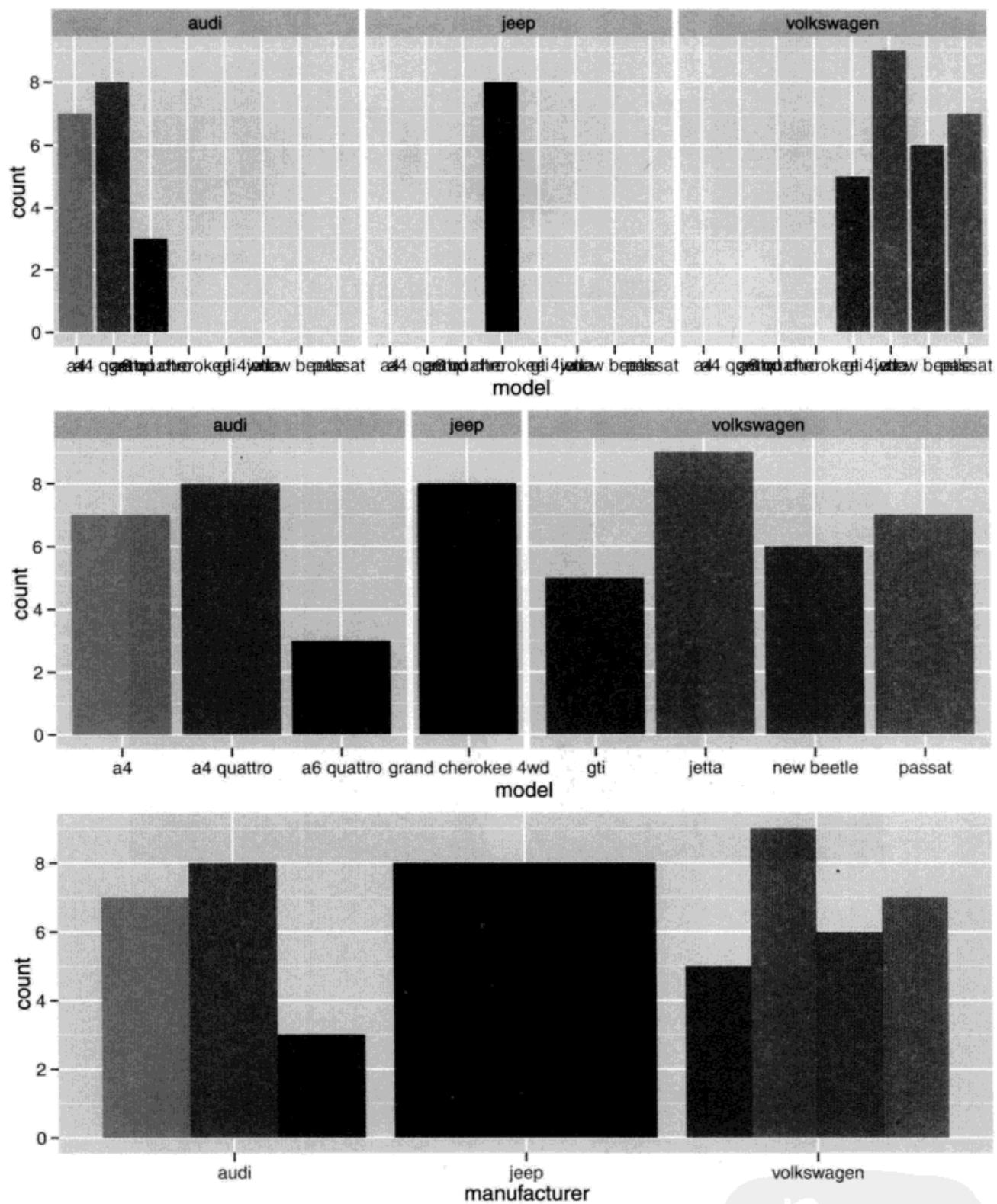
彩图 27 第 121 页, 图 6.14



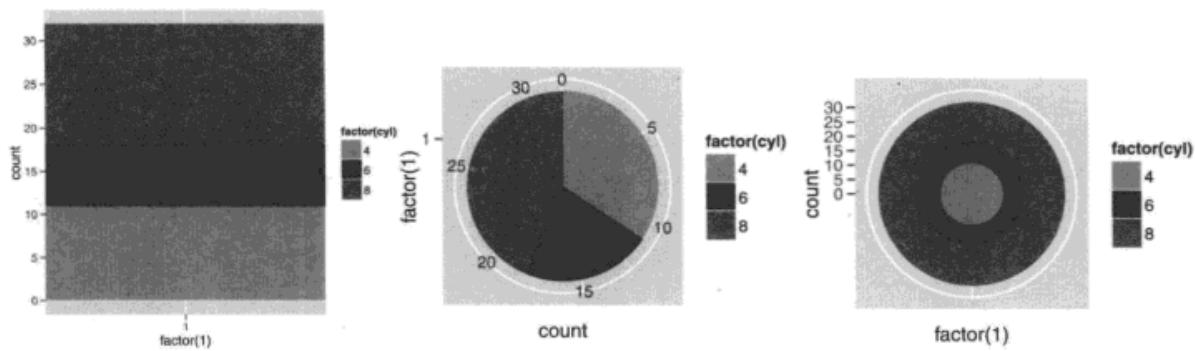
彩图 28 第 134 页, 图 7.7



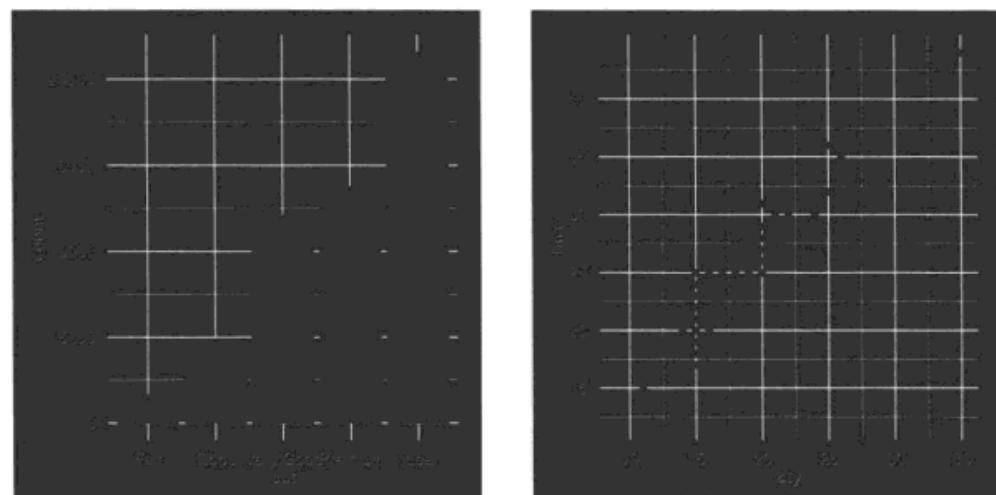
彩图 29 第 135 页, 图 7.8



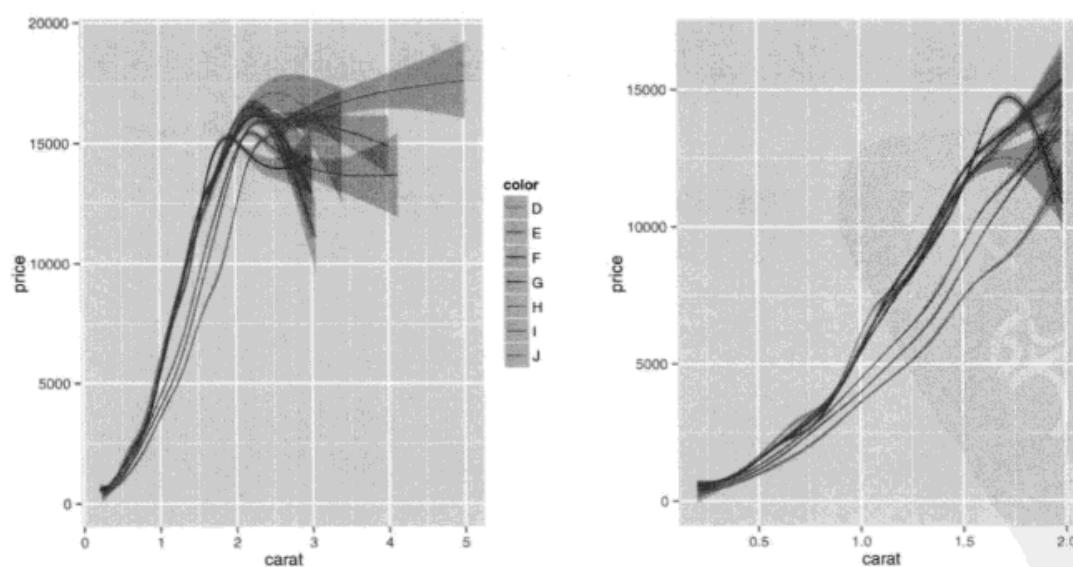
彩图 30 第 137 页, 图 7.9



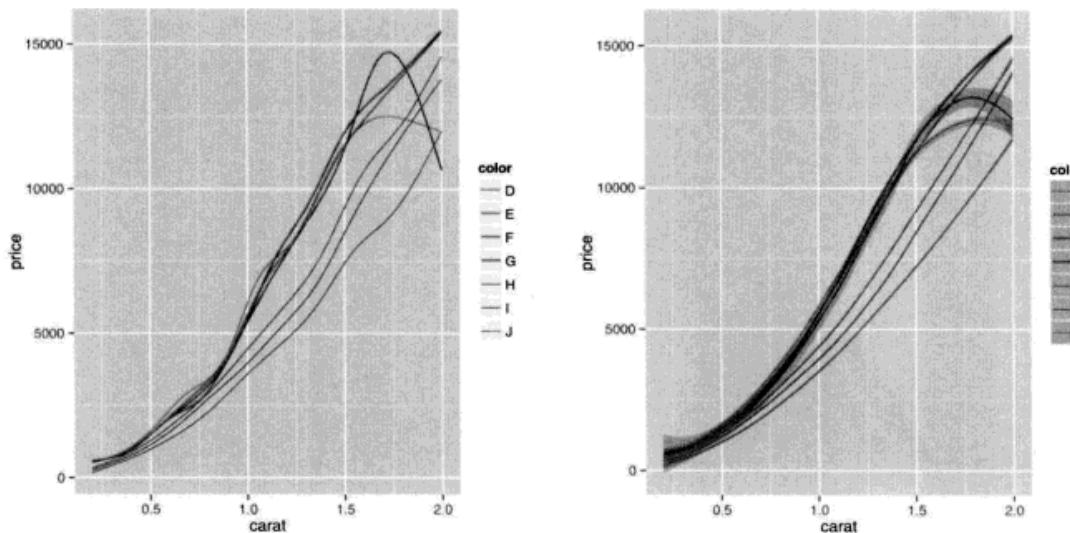
彩图 31 第 145 页, 图 7.17



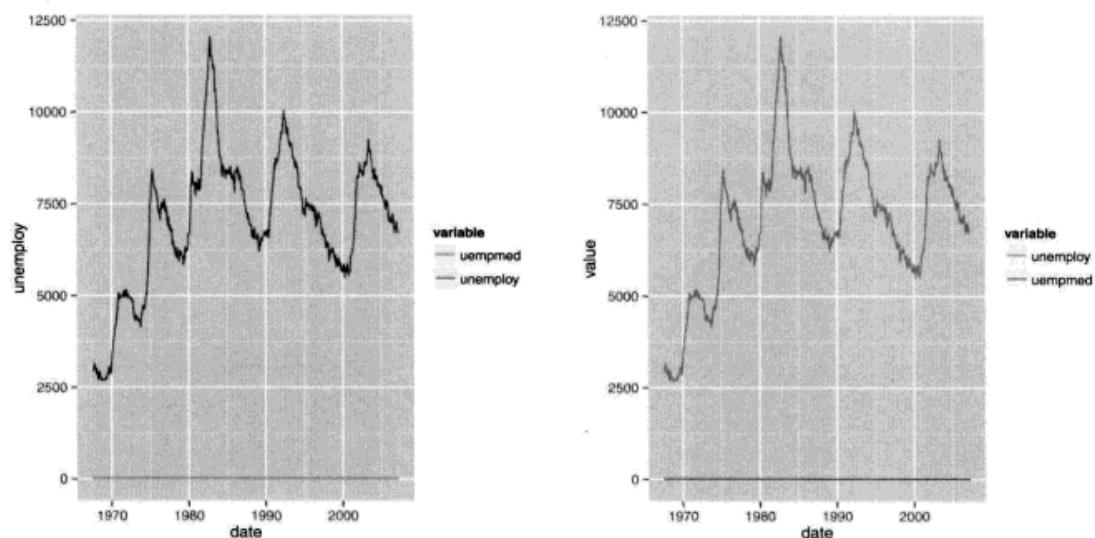
彩图 32 第 155 页, 图 8.6



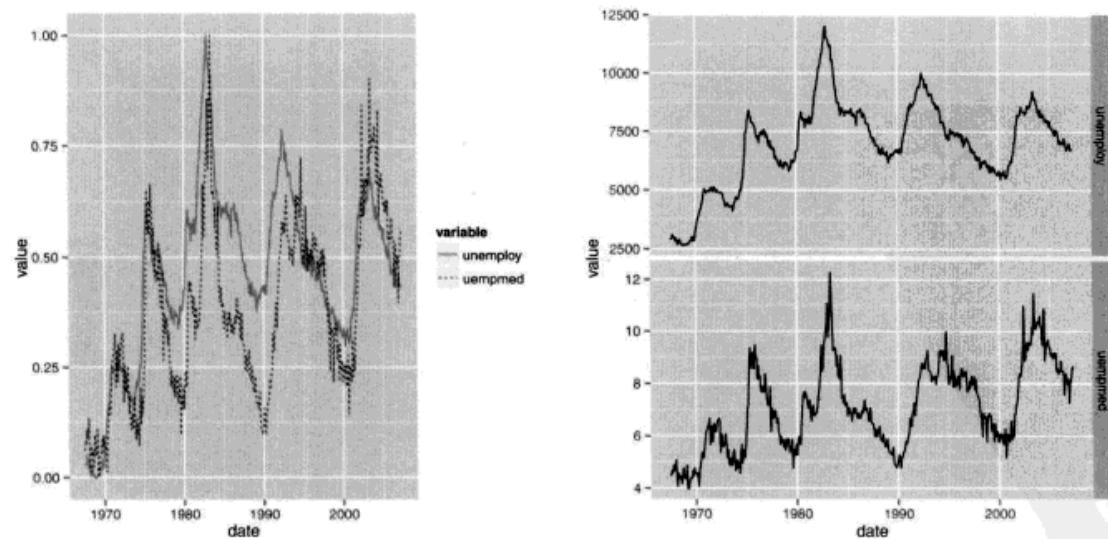
彩图 33 第 169 页, 图 9.1



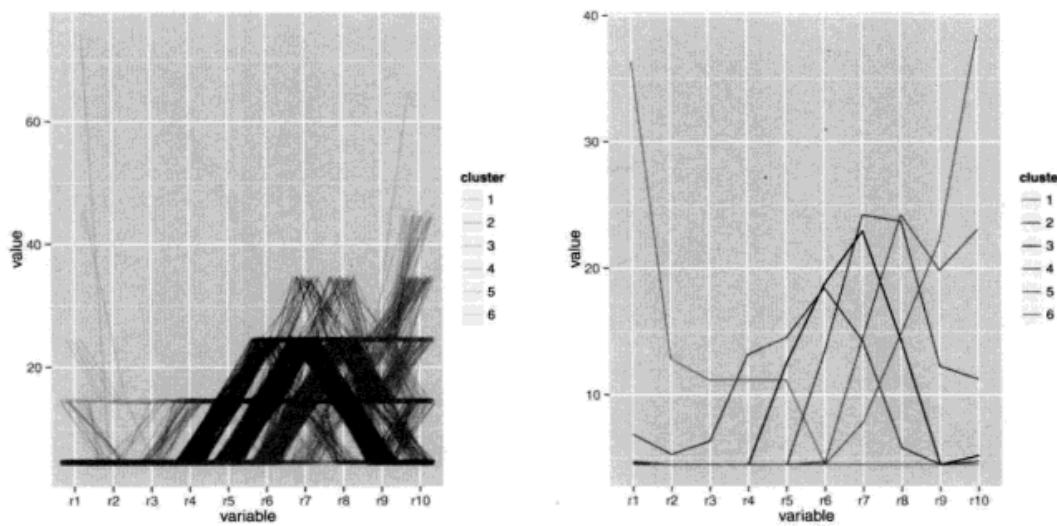
彩图 34 第 170 页, 图 9.2



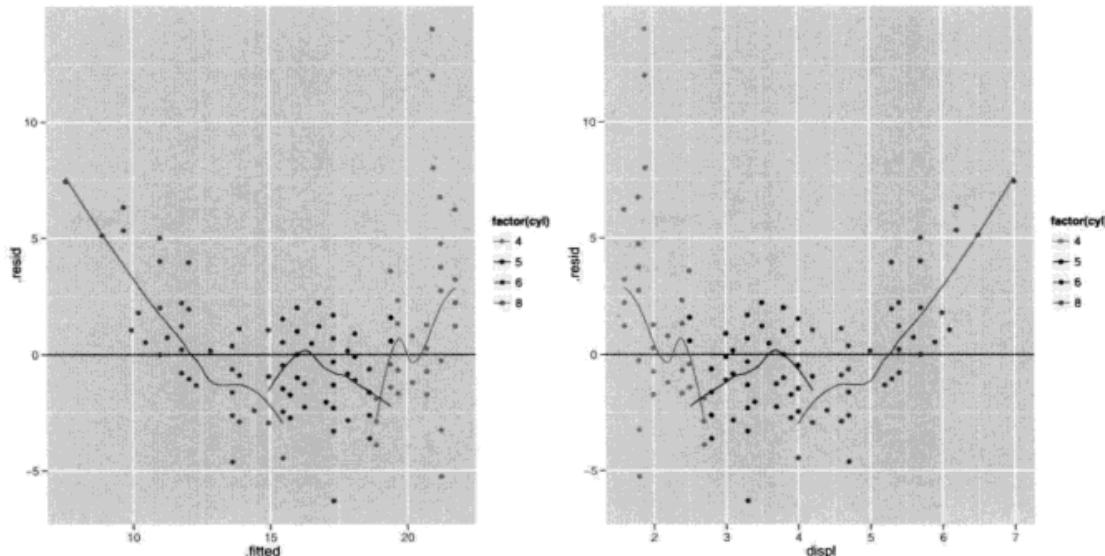
彩图 35 第 174 页, 图 9.4



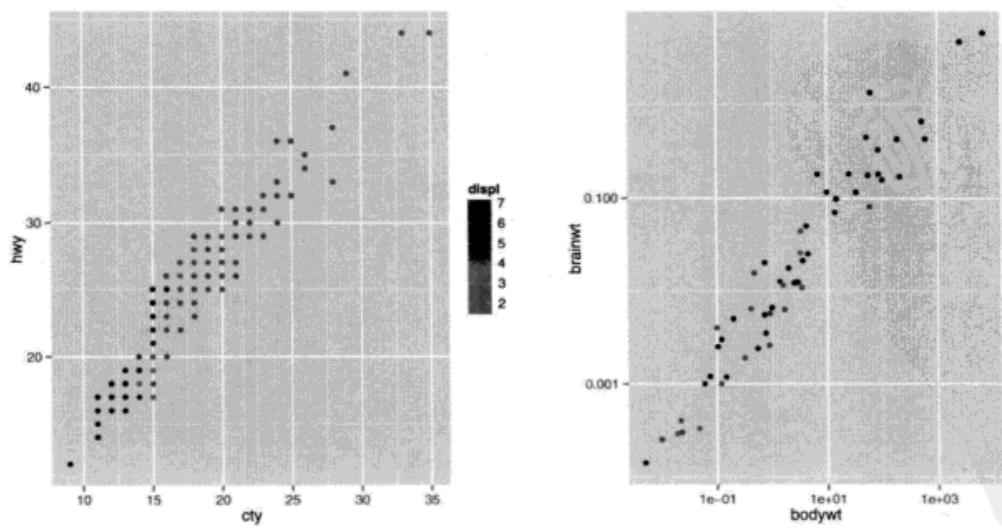
彩图 36 第 174 页, 图 9.5



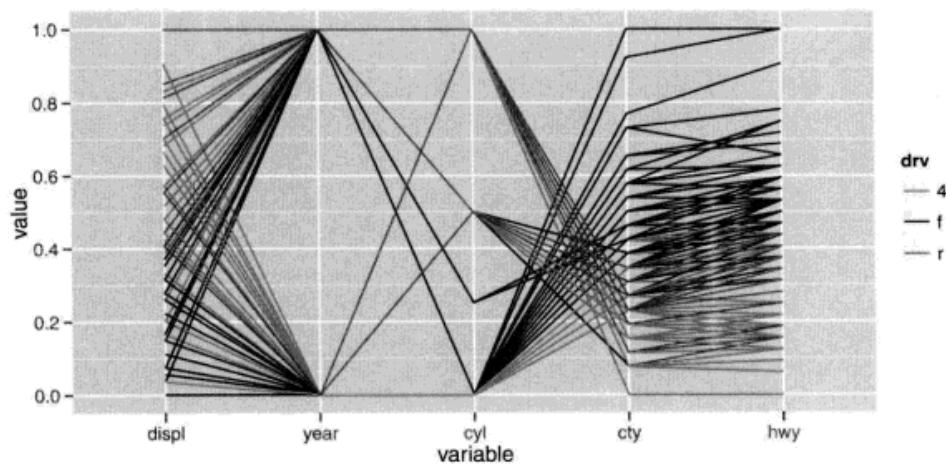
彩图 37 第 177 页, 图 9.7



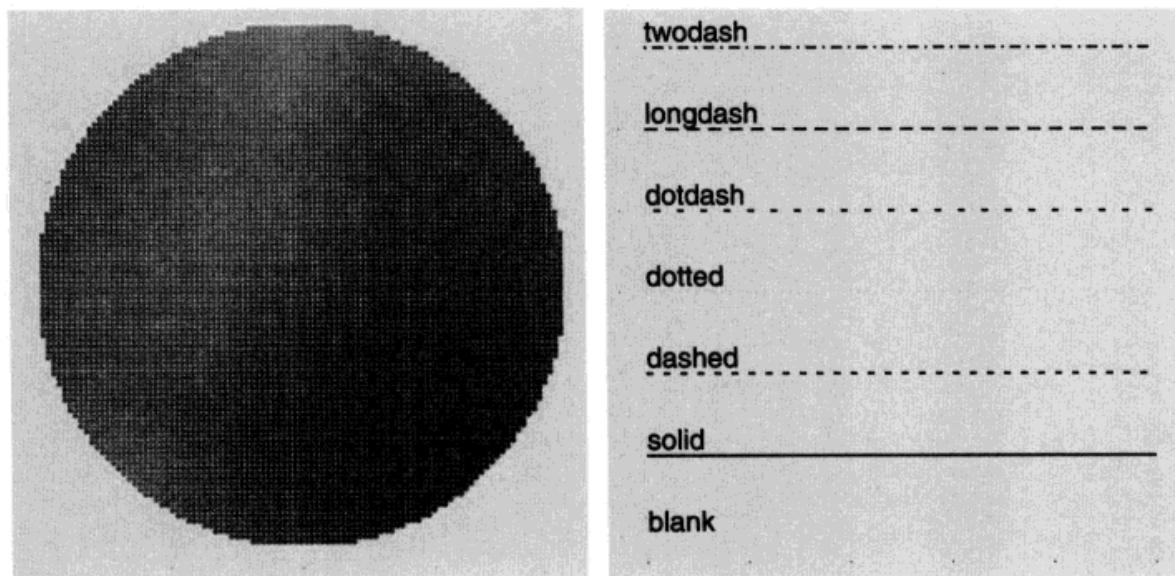
彩图 38 第 183 页, 图 9.12



彩图 39 第 186 页, 图 10.2

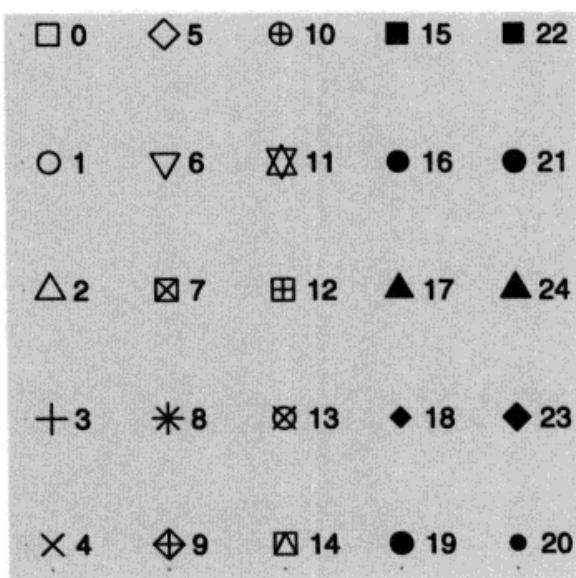


彩图 40 第 189 页

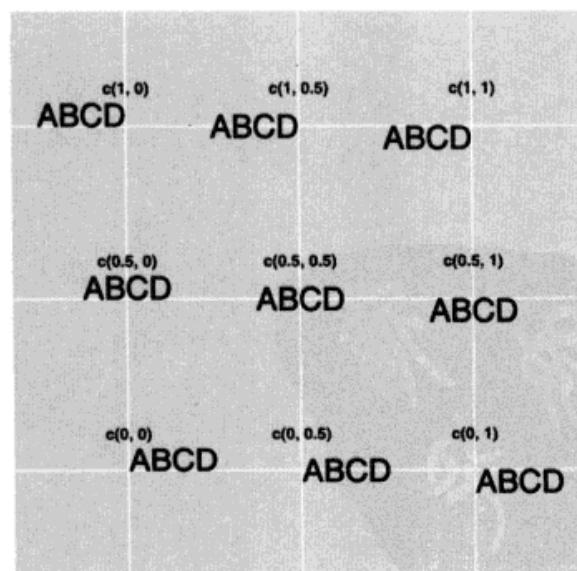


(a) LUV 色系空间中的颜色

(b) 内置的线条类型



(c) R 绘图形状。元素为黑色、背景色为蓝色。
第 25 号符号 (未画出) 可由 24 号符号翻转 180
度得到 (即黑边蓝底倒三角)



(d) 水平、竖直位置的调整

彩图 41 附录第 201 页, 图 B.1