整理数据

（这是 full tidy data paper的非正式和代码繁重的版本。请参阅该文件以获取更多详细信息。）

数据整理

人们常说，80% 的数据分析都花在了清理和准备数据上。这不仅是第一步，而且随着新问题的出现或新数据的收集，它必须在分析过程中多次重复。为了解决这个问题，本文重点介绍了数据清理中一个很小但很重要的方面，我称之为数据**整理**：构建数据集以促进分析。

整理数据的原则提供了一种在数据集中组织数据值的标准方法。标准使初始数据清理更容易，因为您无需每次都从头开始并重新发明轮子。整洁数据标准旨在促进数据的初步探索和分析，并简化协同工作的数据分析工具的开发。当前的工具通常需要翻译。您必须花时间修改一个工具的输出，以便将其输入到另一个工具中。整洁的数据集和整洁的工具相辅相成，使数据分析更容易，让您专注于有趣的领域问题，而不是无趣的数据物流。

定义整洁的数据

> 幸福的家庭都是相似的；每个不幸的家庭都有自己的不幸——列夫·托尔斯泰

像家庭一样，整洁的数据集都是相似的，但每个凌乱的数据集都有自己的凌乱方式。整洁的数据集提供了一种标准化的方式来将数据集的结构（其物理布局）与其语义（其含义）联系起来。在本节中，我将提供一些标准词汇表来描述数据集的结构和语义，然后使用这些定义来定义整洁的数据。

数据结构

大多数统计数据集是由**行**和**列**组成的数据框。列几乎总是被标记，而行有时被标记。以下代码以在野外常见的格式提供了有关虚构教室的一些数据。该表有三列四行，行和列都有标签。

```
classroom <- read.csv("classroom.csv", stringsAsFactors = FALSE)
classroom
#>     name quiz1 quiz2 test1
#> 1  Billy  <NA>     D     C
#> 2   Suzy     F  <NA>  <NA>
#> 3 Lionel     B     C     B
#> 4  Jenny     A     A     B
```

有多种方法可以构建相同的底层数据。下表显示了与上面相同的数据，但行和列已转置。

```
read.csv("classroom2.csv", stringsAsFactors = FALSE)
#>   assessment Billy  Suzy Lionel Jenny
#> 1      quiz1  <NA> FALSE      B     A
#> 2      quiz2     D    NA      C     A
#> 3      test1     C    NA      B     B
```

数据相同，但布局不同。我们的行和列词汇根本不足以描述为什么两个表代表相同的数据。除了外观之外，我们还需要一种方法来描述表中显示的值的底层语义或含义。

数据语义

数据集是一组**值**，通常是数字（如果是定量的）或字符串（如果是定性的）。值以两种方式组织。每个值都属于一个**变量**和一个**观察值**。变量包含跨单位测量相同基础属性（如高度、温度、持续时间）的所有值。一个观察值包含跨属性在同一单位（如一个人、一天或一个种族）上测量的所有值。

课堂数据的整洁版本如下所示：（稍后您将了解这些功能的工作原理）

```
library(tidyr)
library(dplyr)
```

```
classroom2 <- classroom %>%
  pivot_longer(quiz1:test1, names_to = "assessment", values_to = "grade") %>%
  arrange(name, assessment)
classroom2
#> # A tibble: 12 × 3
#>   name  assessment grade
#>   <chr> <chr>      <chr>
#> 1 Billy quiz1      <NA>
#> 2 Billy quiz2      D
#> 3 Billy test1      C
#> 4 Jenny quiz1      A
#> 5 Jenny quiz2      A
#> 6 Jenny test1      B
#> # … with 6 more rows
```

这使得值、变量和观察值更加清晰。该数据集包含 36 个值，代表三个变量和 12 个观测值。变量是：

1. `name`，有四个可能的值（Billy、Suzy、Lionel 和 Jenny）。
2. `assessment`，具有三个可能的值（quiz1、quiz2 和 test1）。
3. `grade`，有五个或六个值，具体取决于您如何看待缺失值（A、B、C、D、F、NA）。

整洁的数据框明确地告诉我们观察的定义。在这个教室，的每一个组合`name`和`assessment`是一个单一的测量观察。数据集还告诉我们缺失值，这些值可以而且确实有意义。比利缺席了第一次测验，但试图挽救他的成绩。苏西第一次考试不及格，所以她决定退课。为了计算比利的最终成绩，我们可以用 F 替换这个缺失值（或者他可能有第二次参加测验的机会）。然而，如果我们想知道测试 1 的类平均值，删除 Suzy 的结构缺失值比输入新值更合适。

对于给定的数据集，通常很容易弄清楚什么是观察值和什么是变量，但通常很难精确定义变量和观察值。例如，如果教室数据中的列是`height`并且`weight`我们很乐意将它们称为变量。如果列是`height`and `width`，那么就不太清楚了，因为我们可能会将高度和宽度视为`dimension`变量的值。如果列是`home phone`and `work phone`，我们可以将它们视为两个变量，但在欺诈检测环境中，我们可能需要变量`phone number`和`number type`因为多人使用一个电话号码可能意味着欺诈。一般的经验法则是，它更容易描述（例如，变量之间的功能关系`z`是线性组合`x`和`y`，`density`是比例`weight`于`volume`）比行之间，并且更容易进行观察的组之间的比较（例如，组 a 的平均值与组 b) 的平均值比列组之间的平均值。

在给定的分析中，可能存在多个观察级别。例如，在新的过敏药物试验中，我们可能有三种观察类型：从每个人收集的人口统计数据（`age`, `sex`, `race`)、每天从每个人收集的医学数据（`number of sneezes`, `redness of eyes`) 和每天收集的气象数据（`temperature`, `pollen count`）。

变量可能会在分析过程中发生变化。原始数据中的变量通常是非常细粒度的，并且可能会增加额外的建模复杂性，以获得很少的解释性收益。例如，许多调查询问同一问题的变体，以更好地了解潜在特征。在分析的早期阶段，变量对应于问题。在后期阶段，您将重点转移到特征上，通过平均多个问题来计算。这大大简化了分析，因为您不需要分层模型，而且您通常可以假装数据是连续的，而不是离散的。

整理数据

整理数据是将数据集的含义映射到其结构的标准方法。数据集是混乱还是整洁取决于行、列和表与观察、变量和类型的匹配方式。在**整齐的数据中**：

1. 每一列都是一个变量。
2. 每一行都是一个观察。
3. 每个单元格都是一个值。

这是 Codd 的第 3 范式，但具有以统计语言构建的约束，并且重点放在单个数据集而不是关系数据库中常见的许多连接数据集上。**杂乱数据**是**数据**的任何其他排列。

整洁的数据使分析师或计算机可以轻松提取所需的变量，因为它提供了构建数据集的标准方法。比较不同版本的课堂数据：在凌乱的版本中你需要使用不同的策略来提取不同的变量。这会减慢分析速度并导致错误。如果您考虑有多少数据分析操作涉及变量（每个聚合函数）中的所有值，您就会发现以简单、标准的方式提取这些值是多么重要。整洁的数据特别适合像 R 这样的向量化编程语言，因为布局确保来自同一观察的不同变量的值总是成对的。

虽然变量和观察的顺序不会影响分析，但良好的排序可以更轻松地扫描原始值。组织变量的一种方法是根据它们在分析中的作用：值是由数据收集的设计确定的，还是在实验过程中测量的？固定变量描述了实验设计并且是预先知道的。计算机科学家通常将固定变量称为维度，而统计学家通常用随机变量的下标来表示它们。测量变量是我们在研究中实际测量的变量。固定变量应该排在第一位，然后是测量变量，每个变量都是有序的，以便相关变量是连续的。然后可以按第一个变量对行进行排序，打破与第二个和后续（固定）变量的联系。

整理凌乱的数据集

真实数据集可以而且经常以几乎所有可以想象的方式违反整洁数据的三个原则。虽然有时您确实会得到一个可以立即开始分析的数据集，但这是例外，而不是规则。本节描述了混乱数据集的五个最常见问题，以及它们的补救措施：

- 列标题是值，而不是变量名称。
- 多个变量存储在一列中。
- 变量存储在行和列中。
- 多个类型的观测单元存储在同一个表中。
- 单个观测单元存储在多个表中。

令人惊讶的是，大多数杂乱的数据集，包括上面没有明确描述的杂乱类型，都可以用一小组工具进行整理：旋转（更长更宽）和分离。以下部分说明了我遇到的真实数据集的每个问题，并展示了如何整理它们。

列标题是值，而不是变量名

一种常见的混乱数据集是为展示而设计的表格数据，其中变量形成行和列，列标题是值，而不是变量名称。虽然我认为这种安排很混乱，但在某些情况下它可能非常有用。它为完全交叉的设计提供有效的存储，如果所需的运算可以表示为矩阵运算，它可以导致极其高效的计算。

以下代码显示了这种形式的典型数据集的一个子集。该数据集探讨了美国收入与宗教之间的关系。它来自皮尤研究中心（Pew Research Center）制作的一份报告，这是一家美国智库，收集有关从宗教到互联网等话题的态度数据，并生成许多包含这种格式数据集的报告。

```
relig_income
#> # A tibble: 18 × 11
#>   religion  `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
#>   <chr>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
#> 1 Agnostic       27        34        60        81        76       137        122
#> 2 Atheist        12        27        37        52        35        70         73
#> 3 Buddhist       27        21        30        34        33        58         62
#> 4 Catholic      418       617       732       670       638      1116        949
#> 5 Don't kn…      15        14        15        11        10        35         21
#> 6 Evangeli…     575       869      1064       982       881      1486        949
```

```
#> # … with 12 more rows, and 3 more variables: $100-150k <dbl>, >150k <dbl>,
#> #   Don't know/refused <dbl>
```

该数据集具有三个变量religion，income和frequency。整理一下，我们需要**枢转**的非可变列成两列的键-值对。此操作通常被描述为使宽数据集更长（或更高）。

在透视变量时，我们需要提供要创建的新键值列的名称。定义要旋转的列（除宗教之外的每一列）后，您将需要关键列的名称，即由列标题的值定义的变量的名称。在这种情况下，它是income. 第二个参数是值列的名称，frequency。

```
relig_income %>%
  pivot_longer(-religion, names_to = "income", values_to = "frequency")
#> # A tibble: 180 × 3
#>   religion income  frequency
#>   <chr>    <chr>       <dbl>
#> 1 Agnostic <$10k          27
#> 2 Agnostic $10-20k        34
#> 3 Agnostic $20-30k        60
#> 4 Agnostic $30-40k        81
#> 5 Agnostic $40-50k        76
#> 6 Agnostic $50-75k       137
#> # … with 174 more rows
```

这种形式是整齐，因为每列表示一个变量，每一行代表一个观测值，在这种情况下对应于组合的人口统计单元religion和income。

这种格式也用于记录一段时间内有规律间隔的观察结果。例如，下面的记录显示的日期一首在Billboard数据集首次进入广告牌前100名。它有一个变量artist，track，date.entered，rank和week。进入前 100 后每周的排名记录在 75 列中，wk1至wk75. 这种存储形式并不整洁，但对于数据录入很有用。它减少了重复，否则每周的每首歌曲都需要自己的行，并且需要重复歌曲元数据，如标题和艺术家。这将在多种类型中进行更深入的讨论。

```
billboard
#> # A tibble: 317 × 79
#>   artist   track    date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
#>   <chr>    <chr>    <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 2 Pac    Baby Do… 2000-02-26      87    82    72    77    87    94    99    NA
#> 2 2Ge+her  The Har… 2000-09-02      91    87    92    NA    NA    NA    NA    NA
#> 3 3 Doors… Krypton… 2000-04-08      81    70    68    67    66    57    54    53
#> 4 3 Doors… Loser    2000-10-21      76    76    72    69    67    65    55    59
#> 5 504 Boyz Wobble … 2000-04-15      57    34    25    17    17    31    36    49
#> 6 98^0     Give Me… 2000-08-19      51    39    34    26    26    19     2     2
#> # … with 311 more rows, and 68 more variables: wk9 <dbl>, wk10 <dbl>,
#> #   wk11 <dbl>, wk12 <dbl>, wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>,
#> #   wk17 <dbl>, wk18 <dbl>, wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>,
#> #   wk23 <dbl>, wk24 <dbl>, wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>,
#> #   wk29 <dbl>, wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>,
#> #   wk35 <dbl>, wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>,
#> #   wk41 <dbl>, wk42 <dbl>, wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, …
```

为了整理这个数据集，我们首先使用pivot_longer()使数据集更长。我们将列从 转换wk1为wk76，为其名称创建一个新列week，并为其值创建一个新值rank：

```
billboard2 <- billboard %>%
  pivot_longer(
    wk1:wk76,
    names_to = "week",
```

```
    values_to = "rank",
    values_drop_na = TRUE
  )
billboard2
#> # A tibble: 5,307 × 5
#>   artist track                   date.entered week   rank
#>   <chr>  <chr>                   <date>       <chr> <dbl>
#> 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1      87
#> 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2      82
#> 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3      72
#> 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4      77
#> 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5      87
#> 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6      94
#> # … with 5,301 more rows
```

在这里，我们使用`values_drop_na = TRUE`从排名列中删除任何缺失值。在此数据中，缺失值表示歌曲不在图表中的周数，因此可以安全地删除。

在这种情况下，做一点清理也很好，将周变量转换为数字，并找出图表上每周对应的日期：

```
billboard3 <- billboard2 %>%
  mutate(
    week = as.integer(gsub("wk", "", week)),
    date = as.Date(date.entered) + 7 * (week - 1),
    date.entered = NULL
  )
billboard3
#> # A tibble: 5,307 × 5
#>   artist track                    week  rank date
#>   <chr>  <chr>                   <int> <dbl> <date>
#> 1 2 Pac  Baby Don't Cry (Keep...     1    87 2000-02-26
#> 2 2 Pac  Baby Don't Cry (Keep...     2    82 2000-03-04
#> 3 2 Pac  Baby Don't Cry (Keep...     3    72 2000-03-11
#> 4 2 Pac  Baby Don't Cry (Keep...     4    77 2000-03-18
#> 5 2 Pac  Baby Don't Cry (Keep...     5    87 2000-03-25
#> 6 2 Pac  Baby Don't Cry (Keep...     6    94 2000-04-01
#> # … with 5,301 more rows
```

最后，对数据进行排序总是一个好主意。我们可以按艺术家、曲目和周来完成：

```
billboard3 %>% arrange(artist, track, week)
#> # A tibble: 5,307 × 5
#>   artist track                    week  rank date
#>   <chr>  <chr>                   <int> <dbl> <date>
#> 1 2 Pac  Baby Don't Cry (Keep...     1    87 2000-02-26
#> 2 2 Pac  Baby Don't Cry (Keep...     2    82 2000-03-04
#> 3 2 Pac  Baby Don't Cry (Keep...     3    72 2000-03-11
#> 4 2 Pac  Baby Don't Cry (Keep...     4    77 2000-03-18
#> 5 2 Pac  Baby Don't Cry (Keep...     5    87 2000-03-25
#> 6 2 Pac  Baby Don't Cry (Keep...     6    94 2000-04-01
#> # … with 5,301 more rows
```

或按日期和排名：

```
billboard3 %>% arrange(date, rank)
#> # A tibble: 5,307 × 5
```

```
#>   artist   track   week  rank date
#>   <chr>    <chr>  <int> <dbl> <date>
#> 1 Lonestar Amazed     1    81 1999-06-05
#> 2 Lonestar Amazed     2    54 1999-06-12
#> 3 Lonestar Amazed     3    44 1999-06-19
#> 4 Lonestar Amazed     4    39 1999-06-26
#> 5 Lonestar Amazed     5    38 1999-07-03
#> 6 Lonestar Amazed     6    33 1999-07-10
#> # … with 5,301 more rows
```

一列存储多个变量

在透视列之后，键列有时是多个底层变量名称的组合。这发生在`tb`（结核病）数据集中，如下所示。该数据集来自世界卫生组织，并通过记录确诊肺结核病例的计数`country`，`year`以及人口群体。人口统计组按`sex`(m, f) 和`age`(0-14, 15-25, 25-34, 35-44, 45-54, 55-64, 未知)细分。

```
tb <- as_tibble(read.csv("tb.csv", stringsAsFactors = FALSE))
tb
#> # A tibble: 5,769 × 22
#>   iso2   year   m04  m514  m014 m1524 m2534 m3544 m4554 m5564   m65    mu   f04
#>   <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#> 1 AD     1989    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 2 AD     1990    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 3 AD     1991    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 4 AD     1992    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 5 AD     1993    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 6 AD     1994    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> # … with 5,763 more rows, and 9 more variables: f514 <int>, f014 <int>,
#> #   f1524 <int>, f2534 <int>, f3544 <int>, f4554 <int>, f5564 <int>, f65 <int>,
#> #   fu <int>
```

首先我们`pivot_longer()`用来收集非变量列：

```
tb2 <- tb %>%
  pivot_longer(
    !c(iso2, year),
    names_to = "demo",
    values_to = "n",
    values_drop_na = TRUE
  )
tb2
#> # A tibble: 35,750 × 4
#>   iso2   year demo      n
#>   <chr> <int> <chr> <int>
#> 1 AD     1996 m014      0
#> 2 AD     1996 m1524     0
#> 3 AD     1996 m2534     0
#> 4 AD     1996 m3544     4
#> 5 AD     1996 m4554     1
#> 6 AD     1996 m5564     0
#> # … with 35,744 more rows
```

这种格式的列标题通常由非字母数字字符（例如.，-，_，:）分隔 ，或者具有固定宽度的格式，就像在这个数据集中一样。`separate()`可以轻松地将复合变量拆分为单个变量。您可以将正则表达式传递给它进行拆分

（默认是在非字母数字列上拆分），也可以传递一个字符位置向量。在这种情况下，我们要在第一个字符后拆分：

```
tb3 <- tb2 %>%
  separate(demo, c("sex", "age"), 1)
tb3
#> # A tibble: 35,750 × 5
#>   iso2   year sex   age       n
#>   <chr> <int> <chr> <chr> <int>
#> 1 AD     1996 m     014       0
#> 2 AD     1996 m     1524      0
#> 3 AD     1996 m     2534      0
#> 4 AD     1996 m     3544      4
#> 5 AD     1996 m     4554      1
#> 6 AD     1996 m     5564      0
#> # … with 35,744 more rows
```

以这种形式存储值解决了原始数据中的问题。我们想比较比率，而不是计数，这意味着我们需要了解人口。在原始格式中，没有简单的方法来添加总体变量。它必须存储在单独的表中，这使得很难将人口与计数正确匹配。以整洁的形式，为人口和比率添加变量很容易，因为它们只是附加列。

在这种情况下，我们还可以通过向 提供多个列名并向`names_to`提供分组的正则表达式来一步完成转换`names_pattern`：

```
tb %>% pivot_longer(
  !c(iso2, year),
  names_to = c("sex", "age"),
  names_pattern = "(.)(.+)",
  values_to = "n",
  values_drop_na = TRUE
)
#> # A tibble: 35,750 × 5
#>   iso2   year sex   age       n
#>   <chr> <int> <chr> <chr> <int>
#> 1 AD     1996 m     014       0
#> 2 AD     1996 m     1524      0
#> 3 AD     1996 m     2534      0
#> 4 AD     1996 m     3544      4
#> 5 AD     1996 m     4554      1
#> 6 AD     1996 m     5564      0
#> # … with 35,744 more rows
```

变量同时存储在行和列中

当变量同时存储在行和列中时，会出现最复杂形式的混乱数据。下面的代码从全球历史气候学网络加载墨西哥一个气象站 (MX17004) 2010 年五个月的每日天气数据。

```
weather <- as_tibble(read.csv("weather.csv", stringsAsFactors = FALSE))
weather
#> # A tibble: 22 × 35
#>   id       year month element    d1    d2    d3    d4    d5    d6    d7    d8
#>   <chr>   <int> <int> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 MX17004  2010     1 tmax       NA    NA    NA    NA    NA    NA    NA    NA
#> 2 MX17004  2010     1 tmin       NA    NA    NA    NA    NA    NA    NA    NA
#> 3 MX17004  2010     2 tmax       NA  27.3  24.1    NA    NA    NA    NA    NA
```

```
#> 4 MX17004  2010     2 tmin       NA  14.4  14.4    NA  NA      NA    NA    NA
#> 5 MX17004  2010     3 tmax       NA  NA    NA      NA  32.1    NA    NA    NA
#> 6 MX17004  2010     3 tmin       NA  NA    NA      NA  14.2    NA    NA    NA
#> # … with 16 more rows, and 23 more variables: d9 <lgl>, d10 <dbl>, d11 <dbl>,
#> #   d12 <lgl>, d13 <dbl>, d14 <dbl>, d15 <dbl>, d16 <dbl>, d17 <dbl>,
#> #   d18 <lgl>, d19 <lgl>, d20 <lgl>, d21 <lgl>, d22 <lgl>, d23 <dbl>,
#> #   d24 <lgl>, d25 <dbl>, d26 <dbl>, d27 <dbl>, d28 <dbl>, d29 <dbl>,
#> #   d30 <dbl>, d31 <dbl>
```

它在单个列 (`id`, `year`, `month`) 中具有变量，分布在列 (`day`, d1-d31) 和行 (`tmin`, `tmax`) (最低和最高温度) 中。少于 31 天的月份具有该月最后一天的结构性缺失值。

为了整理这个数据集，我们首先使用 pivot_longer 来收集日期列：

```
weather2 <- weather %>%
  pivot_longer(
    d1:d31,
    names_to = "day",
    values_to = "value",
    values_drop_na = TRUE
  )
weather2
#> # A tibble: 66 × 6
#>   id       year month element day   value
#>   <chr>   <int> <int> <chr>   <chr> <dbl>
#> 1 MX17004  2010     1 tmax    d30    27.8
#> 2 MX17004  2010     1 tmin    d30    14.5
#> 3 MX17004  2010     2 tmax    d2     27.3
#> 4 MX17004  2010     2 tmax    d3     24.1
#> 5 MX17004  2010     2 tmax    d11    29.7
#> 6 MX17004  2010     2 tmax    d23    29.9
#> # … with 60 more rows
```

为了演示，我删除了缺失值，使它们隐式而不是显式。这是可以的，因为我们知道每个月有多少天，并且可以轻松重建显式缺失值。

我们也会做一点清洁：

```
weather3 <- weather2 %>%
  mutate(day = as.integer(gsub("d", "", day))) %>%
  select(id, year, month, day, element, value)
weather3
#> # A tibble: 66 × 6
#>   id       year month   day element value
#>   <chr>   <int> <int> <int> <chr>   <dbl>
#> 1 MX17004  2010     1    30 tmax     27.8
#> 2 MX17004  2010     1    30 tmin     14.5
#> 3 MX17004  2010     2     2 tmax     27.3
#> 4 MX17004  2010     2     3 tmax     24.1
#> 5 MX17004  2010     2    11 tmax     29.7
#> 6 MX17004  2010     2    23 tmax     29.9
#> # … with 60 more rows
```

这个数据集大多是整洁的，但`element`列不是变量；它存储变量的名称。（本例中未显示其他气象变量`prcp`（降水）和`snow`（降雪））。解决这个问题需要拓宽数据：`pivot_wider()`与`pivot_longer()`，旋转`element`并`value`在多列之间退出：

```
weather3 %>%
  pivot_wider(names_from = element, values_from = value)
#> # A tibble: 33 × 6
#>   id       year month   day  tmax  tmin
#>   <chr>   <int> <int> <int> <dbl> <dbl>
#> 1 MX17004  2010     1    30  27.8  14.5
#> 2 MX17004  2010     2     2  27.3  14.4
#> 3 MX17004  2010     2     3  24.1  14.4
#> 4 MX17004  2010     2    11  29.7  13.4
#> 5 MX17004  2010     2    23  29.9  10.7
#> 6 MX17004  2010     3     5  32.1  14.2
#> # … with 27 more rows
```

这种形式很整洁：每一列有一个变量，每一行代表一天。

一张表中的多种类型

数据集通常涉及在不同类型的观测单位的多个级别收集的值。在整理期间，每种类型的观察单元都应存储在自己的表中。这与数据库规范化的思想密切相关，其中每个事实只在一个地方表达。这很重要，否则可能会出现不一致的情况。

广告牌数据集实际上包含对两种观察单位的观察：歌曲及其每周排名。这通过关于歌曲的重复事实表现出来：`artist`重复了很多次。

此数据集需要被分解成两个部分：一首歌的数据集，其存储`artist`和`song name`，以及排名数据集这给`rank`的`song`每个`week`。我们首先提取一个`song`数据集：

```
song <- billboard3 %>%
  distinct(artist, track) %>%
  mutate(song_id = row_number())
song
#> # A tibble: 317 × 3
#>   artist       track                   song_id
#>   <chr>        <chr>                     <int>
#> 1 2 Pac        Baby Don't Cry (Keep...       1
#> 2 2Ge+her      The Hardest Part Of ...       2
#> 3 3 Doors Down Kryptonite                   3
#> 4 3 Doors Down Loser                        4
#> 5 504 Boyz     Wobble Wobble                5
#> 6 98^0         Give Me Just One Nig...      6
#> # … with 311 more rows
```

然后`rank`通过用指向歌曲详细信息的指针（唯一的歌曲 ID）替换重复的歌曲事实来使用它来制作数据集：

```
rank <- billboard3 %>%
  left_join(song, c("artist", "track")) %>%
  select(song_id, date, week, rank)
rank
#> # A tibble: 5,307 × 4
#>   song_id date        week  rank
#>     <int> <date>     <int> <dbl>
#> 1       1 2000-02-26     1    87
#> 2       1 2000-03-04     2    82
#> 3       1 2000-03-11     3    72
#> 4       1 2000-03-18     4    77
#> 5       1 2000-03-25     5    87
```

```
#> 6       1 2000-04-01     6    94
#> # … with 5,301 more rows
```

你也可以想象一个week数据集，它会记录一周的背景信息，可能是销售的歌曲总数或类似的“人口统计”信息。

标准化对于整理和消除不一致很有用。然而，直接处理关系数据的数据分析工具很少，因此分析通常还需要非规范化或将数据集合并回一张表。

多表中的一种

查找关于分布在多个表或文件中的单一类型观测单元的数据值也很常见。这些表和文件通常由另一个变量拆分，因此每个都代表一个年份、一个人或一个位置。只要单个记录的格式是一致的，这是一个很容易解决的问题：

1. 将文件读入表列表。
2. 对于每个表，添加一个记录原始文件名的新列（文件名通常是一个重要变量的值）。
3. 将所有表合并为一个表。

Purrr 在 R 中使这变得简单。以下代码在目录（data/）中生成一个文件名向量，该向量与正则表达式（以 结尾）相匹配.csv。接下来我们用文件名命名向量的每个元素。我们这样做是因为将在以下步骤中保留名称，确保最终数据框中的每一行都标有其来源。最后，map_dfr()循环遍历每个路径，读取 csv 文件并将结果组合到一个数据框中。

```
library(purrr)
paths <- dir("data", pattern = "\\.csv$", full.names = TRUE)
names(paths) <- basename(paths)
map_dfr(paths, read.csv, stringsAsFactors = FALSE, .id = "filename")
```

一旦您拥有一张桌子，您就可以根据需要执行额外的整理工作。此类清洁的示例可在https://github.com/hadley/data-baby-names中找到，该示例将美国社会保障管理局提供的 129 个年度婴儿姓名表合并到一个文件中。

当数据集结构随时间变化时，会出现更复杂的情况。例如，数据集可能包含不同的变量、具有不同名称的相同变量、不同的文件格式或不同的缺失值约定。这可能需要您单独整理每个文件（或者，如果幸运的话，可以分成小组），然后在整理后将它们组合起来。https://github.com/hadley/data-fuel-economy中展示了这种类型的整理示例，其中显示了1978 年至 2008 年 50,000 多辆汽车的EPA燃油经济性数据的整理。原始数据可在线获取，但每年都存储在一个单独的文件中，并且有四种主要格式，有许多细微的变化，这使得整理这个数据集成为一个相当大的挑战。