Tidy data

2015-09-09

Перевод

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

(Это неформальная и более богатая примерами кода версия публикации. Больше подробностей вы можете найти в ней.)

Придание данным аккуратного вида

Часто говорят, что 80% анализа данных составляет очистка и подготовка данных. И это не просто первый этап, напротив, очистка данных должна повторяться много раз в процессе анализа по мере выявления новых проблем или сбора новых данных. Чтобы помочь справиться с этой задачей, в данной статье рассматривается маленький, но важный аспект очистки данных, который я называю **приданием аккуратного вида** (“tidy” далее перевожу как “аккуратные” - прим. пер.): структурирование наборов данных для облегчения анализа. Принципы аккуратных данных обеспечивают стандартный способ организации значений в наборе данных. Стандарт делает очистку исходных данных проще, поскольку вам не нужно каждый раз начинать с нуля и изобретать велосипед. Стандарт аккуратных данных был разработан, чтобы облегчить первичный разведочный анализ данных и упростить разработку средств для анализа данных, которые хорошо работают в сочетании друг с другом. Имеющиеся инструменты часто требуют перевода [из одного формата в другой]. Вам приходится тратить время на переформатирование вывода одного из них, чтобы затем подать на вход другого. Аккуратные наборы данных и соответствующие инструменты работают плечом к плечу, чтобы сделать анализ данных проще, позволяя вам сконцентрироваться на интересной проблеме в своей предметной области, а не на малоинтересной логистике данных.

Определение аккуратных данных

«Все счастливые семьи похожи друг на друга, каждая несчастливая семья несчастлива по-своему.» (Л. Толстой)
Подобно семьям, все аккуратные наборы данных похожи друг на друга, но каждый беспорядочный набор данных беспорядочен по-своему. Аккуратные наборы данных обеспечивают стандартный способ связи структуры набора данных (его физической компоновки) с семантикой (его значением). В этом разделе я приведу некий стандартный словарь для описания структуры и семантики набора данных, а затем использую эти термины для определения понятия аккуратных данных.

Структура данных

Большинство статистических наборов данных являются таблицами, состоящими из **строк** и **столбцов**. Столбцы имеют метки почти всегда, строки - иногда. Следующий код представляет некоторые данные о воображаемом эксперименте в формате, который обычно можно наблюдать в реальных условиях. Таблица содержит два столбца и три строки; и строки, и столбцы имеют метки (первый столбец посчитан как метки строк - прим. пер.).

```
preg <- read.csv("preg.csv", stringsAsFactors = FALSE)
preg
#>           name treatmenta treatmentb
#> 1   John Smith         NA         18
#> 2     Jane Doe          4          1
#> 3 Mary Johnson          6          7
```

Существует много способов структурирования тех же данных. Следующая таблица показывает те же данные, что и предыдущая, но строки и столбцы были транспонированы.

```
read.csv("preg2.csv", stringsAsFactors = FALSE)
#>   treatment John.Smith Jane.Doe Mary.Johnson
#> 1         a         NA        4            6
#> 2         b         18        1            7
```

Данные те же, но их формат отличается. Наш словарь, состоящий из строк и столбцов, просто недостаточно богат для описания того, почему эти две таблицы представляют одни и те же данные. В дополнение к внешнему виду, мы также нуждаемся в способе описания лежащей в основе семантики, иными словами, значения данных, отображенных в таблице.

Семантика данных

Набор данных является совокупностью **значений**, обычно представленных числами (количественные данные) или строками (качественные/категориальные данные). Значения организованы в двух направлениях. Каждое значение принадлежит **переменной** и **наблюдению**. Переменная содержит все значения, которые отображают один и тот же атрибут (такой как высота, температура, продолжительность) всех объектов. Наблюдение содержит все значения атрибутов, измеренные для одного и того же объекта (такого как человек, день или раса). Аккуратная версия данных о беременности выглядит следующим образом (немного позже вы узнаете, как работают эти функции):

```
library(tidyr)
library(dplyr)
preg2 <- preg %>%
  gather(treatment, n, treatmenta:treatmentb) %>%
  mutate(treatment = gsub("treatment", "", treatment)) %>%
  arrange(name, treatment)
preg2
#>           name treatment  n
#> 1     Jane Doe         a  4
#> 2     Jane Doe         b  1
#> 3   John Smith         a NA
#> 4   John Smith         b 18
#> 5 Mary Johnson         a  6
#> 6 Mary Johnson         b  7
```

Это делает значения, переменные и наблюдения более понятными. Набор данных содержит 18 значений, представляющий три переменные и шесть наблюдений. Переменные:

1. `name` , с тремя возможными значениями (John, Mary и Jane).
2. `treatment` , с двумя возможными значениями (a и b).
3. `n` , с пятью или шестью значениями в зависимости от того, как учитывать пропущенные значения (1, 4, 6, 7, 18, NA).

Дизайн эксперимента дает нам больше информации о структуре наблюдений. В этом эксперименте измерялась каждая комбинация имени и лечения (полный перекрестный дизайн). Дизайн эксперимента также определяет, можно ли безболезненно удалять пропущенные значения. В данном эксперименте пропущенные значения представляют наблюдения, которые должны были быть сделаны, но фактически выполнены не были, поэтому важно сохранить их. Могут быть удалены структурные пропущенные значения, представляющие измерения, которые не могли быть выполнены (например, количество беременных мужчин).
Для конкретного набора данных обычно легко понять, что из себя представляют наблюдения и переменные, но удивительно сложно дать точные определения переменным и наблюдениям как таковым. Например, если столбцами в наборе данных о беременности были бы `height` и `weight`, мы с удовольствием назвали бы их переменными. Если бы столбцами были `height` и `width`, это были бы не столь очевидно, так как мы можем думать о высоте и ширине как о значениях переменной `dimension`. Если бы столбцами были `home phone` и `work phone`, мы могли бы считать их двумя переменными, но в контексте выявления случаев мошенничества нам могли бы понадобиться переменные `phone number` и `number type`, поскольку использование одного телефонного номера многими людьми может свидетельствовать о мошенничестве.
Общее эмпирическое правило заключается в том, что проще описать функциональную взаимосвязь между переменными (например, `z` является линейной комбинацией `x` и `y`, `density` является отношением `weight` к `volume`), чем между строками/наблюдениями, и проще выполнять сравнения между группами наблюдений (например, сравнить среднее группы a со средним группы b), чем между группами переменных.
В отдельном анализе может быть несколько уровней наблюдений. Например, в испытании нового лекарства от аллергии мы можем иметь наблюдения трех типов: демографические данные, собранные для каждого человека (`age`, `sex`, `race`); данные, собранные для каждого человека за каждый день (`number of sneezes`, `redness of eyes`); метеорологические данные, собранные за каждый день (`temperature`, `pollen count`).
Переменные могут изменяться в ходе анализа. Часто переменные в исходных данных слишком детализированные, что может увеличивать сложность моделирования при незначительном улучшении объясняющей способности. Например, при проведении многих опросов задают несколько вариантов одного и того же вопроса для получения лучшей характеристики лежащего в основе изучаемого признака. На ранних стадиях анализа переменные соответствуют вопросам. На поздних стадиях вы концентрируетесь на признаках, вычисленных путем усреднения нескольких вопросов. Это значительно упрощает анализ, так как вам не нужна иерархическая модель, и вы часто можете обрабатывать данные как непрерывные, а не как дискретные.

Аккуратные данные

Аккуратные данные являются стандартным способом отображения содержания набора данных в его структуре. Набор данных является беспорядочным или аккуратным в зависимости от того, как строки, стролбцы и таблицы сопоставляются с наблюдениями, переменными и типами. В аккуратных данных:

1. Каждая переменная формирует столбец.
2. Каждое наблюдение формирует строку.
3. Каждый тип единиц наблюдения формирует таблицу.

Это третья нормальная форма Кодда, но с ограничениями в контексте статистического языка; в центре внимания находится отдельный набор данных вместо многих связанных между собой наборов, как это обычно имеет место в реляционных базах данных. **Беспорядочными данными**являются любые другие компоновки данных. Аккуратные данные делают извлечение нужных переменных более простым для аналитика или для компьютера, поскольку обеспечивается стандартный способ

структурирования набора данных. Сравните разные версии данных о беременности: в беспорядочной версии вы должны использовать разные способы для извлечения разных переменных. Это замедляет анализ и провоцирует ошибки. Если вы посчитаете, как много операций анализа данных включают все значения переменной (каждая агрегирующая функция), то увидите, как важно извлекать эти значения простым стандартным способом. Аккуратные данные особенно хорошо подходят для векторизированных языков программирования, таких как R, потому что такая структура гарантирует, что значения разных переменных для одного и того же наблюдения всегда соответствуют друг другу.
Несмотря на то, что порядок переменных и наблюдений не влияет на анализ, правильное упорядочивание облегчает просмотр исходных значений. Один из способов организации переменных - по их роли в анализе: являются ли значения фиксированными в соответствии с дизайном сбора данных, или же они измеряются в ходе эксперимента? Фиксированные переменные описывают дизайн эксперимента и известны заранее. Специалисты в области компьютерных наук часто называют фиксированные переменные размерностями, а статистики обычно обозначают их с подстрочными индексами по случайным переменным. Измеряемые переменные - это то, что мы на самом деле измеряем в ходе исследования. Фиксированные переменные должны идти первыми, после них - измеряемые переменные, упорядоченные так, чтобы связанные переменные были смежными. Строки затем могут быть упорядочены по первой переменной, а затем по следующим (фиксированным) переменным. Данное соглашение применяется для всех таблиц, приведенных в этой публикации.

Придание аккуратного вида беспорядочным данным

Реальные наборы данных могут нарушать (и часто нарушают) три принципа аккуратных данных практически всеми мыслимыми способами. И хотя вы иногда получаете набор данных, с которым можете немедленно начать анализ, это скорее исключение, чем правило. Этот раздел описывает пять наиболее распространенных проблем беспорядочных наборов данных, а также способы их устранения:

- Заголовки столбцов являются значениями, а не именами переменных
- Несколько переменных хранятся в одном столбце.
- Переменные хранятся и в строках, и в столбцах.
- Несколько типов единиц наблюдения хранятся в одной таблице.
- Одна единица наблюдения хранится в нескольких таблицах.

Удивительно, но большинство беспорядочных наборов данных, включая их виды, не описанные в явном виде выше, могут быть приведены к аккуратному виду с помощью небольшого набора инструментов: сборка, разделения и распределения. Следующие разделы иллюстрируют каждую из проблем реальных наборов данных, с которыми я сталкивался, и показывают, как можно привести их в порядок.

Заголовки столбцов являются значениями, а не именами переменных

Распространенным типом беспорядочных данных являются табличные данные, подготовленные для презентаций, в которых переменные образуют как строки, так и столбцы, и заголовки столбцов являются значениями, а не именами переменных. Хотя я называю такую компоновку беспорядочной, в некоторых случаях она может быть исключительно полезной. Она обеспечивает эффективное хранение для полных перекрестных дизайнов, и это может приводить к чрезвычайно эффективным вычислениям, если требуемые операции могут быть выражены как операции над матрицами.

Следующий код демонстрирует поднабор из типичного набора данных в такой форме. Этот набор данных исследует взаимосвязь между доходом и религией в США. Источником является отчет, созданный Pew Research Center, американским аналитическим центром, который собирает данные, касающиеся различных тем - от религии до интернета, а также создает много отчетов с наборами данных в таком формате.

```
pew <- tbl_df(read.csv("pew.csv", stringsAsFactors = FALSE, check.names = FALSE
))
pew
#> Source: local data frame [18 x 11]
#>
#>                  religion <$10k $10-20k $20-30k $30-40k $40-50k $50-75k
#>                     (chr) (int)   (int)   (int)   (int)   (int)   (int)
#> 1                Agnostic    27      34      60      81      76     137
#> 2                 Atheist    12      27      37      52      35      70
#> 3                Buddhist    27      21      30      34      33      58
#> 4                Catholic   418     617     732     670     638    1116
#> 5       Don’t know/refused    15      14      15      11      10      35
#> 6         Evangelical Prot   575     869    1064     982     881    1486
#> 7                   Hindu     1       9       7       9      11      34
#> 8  Historically Black Prot   228     244     236     238     197     223
#> 9        Jehovah's Witness    20      27      24      24      21      30
#> 10                 Jewish    19      19      25      25      30      95
#> ..                    ...   ...     ...     ...     ...     ...     ...
#> Variables not shown: $75-100k (int), $100-150k (int), >150k (int), Don't
#>   know/refused (int)
```

Этот набор данных содержит три переменные: `religion`, `income` и `frequency`. Чтобы сделать данные аккуратными, мы должны **собрать**столбцы, не являющиеся переменными, в пары ключ-значение, поместив их в два столбца. Это действие часто описывают как перевод широкого набора данных в длинную (или высокую) форму, но я буду избегать этих терминов из-за их неточности.
При сборке переменных мы должны задать имена создаваемых столбцов с ключами и значениями. Первым аргументом является имя столбца с ключами, т.е. имя переменной, определяемой значениями из заголовков [исходных] столбцов. В данном случае, это `income`. Второй аргумент - это имя столбца со значениями, `frequency`. Третий аргумент определяет столбцы для сборки, в данном случае - все столбцы, кроме `religion`.

```
pew %>%
  gather(income, frequency, -religion)
#> Source: local data frame [180 x 3]
#>
#>                  religion income frequency
#>                     (chr) (fctr)     (int)
#> 1                Agnostic  <$10k        27
#> 2                 Atheist  <$10k        12
#> 3                Buddhist  <$10k        27
#> 4                Catholic  <$10k       418
#> 5       Don’t know/refused  <$10k        15
#> 6         Evangelical Prot  <$10k       575
#> 7                   Hindu  <$10k         1
#> 8  Historically Black Prot  <$10k       228
#> 9        Jehovah's Witness  <$10k        20
#> 10                 Jewish  <$10k        19
#> ..                    ...    ...       ...
```

Эта форма является аккуратной, поскольку каждый столбец представляет переменную, а каждая строка - это наблюдение, в данном случае - демографическая единица, соответствующая комбинации `religion` и `income`.
Этот формат также используется для представления наблюдений, равномерно распределенных по времени. Например, набор данных `billboard`, показанный ниже, содержит даты когда песня впервые попала в Топ 100. Он содержит переменные `artist`, `track`, `date.entered`, `rank` и `week`. Места, занимаемые в течение каждой недели после попадания в Топ 100, преставлены в 75 столбцах - от `wk1` до `wk75`. Эта форма хранения не является аккуратной, но она удобна для ввода данных: уменьшается дублирование, поскольку иначе для каждой песни за каждую неделю потребуется отдельная строка, и метаданные песни, такие как название и исполнитель, будут дублироваться. Этот вопрос будет обсуждаться подробнее в разделе про хранение нескольких типов единиц наблюдения.

```
billboard <- tbl_df(read.csv("billboard.csv", stringsAsFactors = FALSE))
billboard
#> Source: local data frame [317 x 81]
#>
#>     year         artist                   track  time date.entered   wk1
#>    (int)          (chr)                   (chr) (chr)        (chr) (int)
#> 1   2000          2 Pac Baby Don't Cry (Keep...  4:22   2000-02-26    87
#> 2   2000        2Ge+her The Hardest Part Of ...  3:15   2000-09-02    91
#> 3   2000   3 Doors Down              Kryptonite  3:53   2000-04-08    81
#> 4   2000   3 Doors Down                   Loser  4:24   2000-10-21    76
#> 5   2000       504 Boyz           Wobble Wobble  3:35   2000-04-15    57
#> 6   2000           98^0 Give Me Just One Nig...  3:24   2000-08-19    51
#> 7   2000        A*Teens           Dancing Queen  3:44   2000-07-08    97
#> 8   2000        Aaliyah           I Don't Wanna  4:15   2000-01-29    84
#> 9   2000        Aaliyah               Try Again  4:03   2000-03-18    59
#> 10  2000 Adams, Yolanda           Open My Heart  5:30   2000-08-26    76
#> ..   ...            ...                     ...   ...          ...   ...
#> Variables not shown: wk2 (int), wk3 (int), wk4 (int), wk5 (int), wk6
#>   (int), wk7 (int), wk8 (int), wk9 (int), wk10 (int), wk11 (int), wk12
#>   (int), wk13 (int), wk14 (int), wk15 (int), wk16 (int), wk17 (int), wk18
#>   (int), wk19 (int), wk20 (int), wk21 (int), wk22 (int), wk23 (int), wk24
#>   (int), wk25 (int), wk26 (int), wk27 (int), wk28 (int), wk29 (int), wk30
#>   (int), wk31 (int), wk32 (int), wk33 (int), wk34 (int), wk35 (int), wk36
#>   (int), wk37 (int), wk38 (int), wk39 (int), wk40 (int), wk41 (int), wk42
#>   (int), wk43 (int), wk44 (int), wk45 (int), wk46 (int), wk47 (int), wk48
#>   (int), wk49 (int), wk50 (int), wk51 (int), wk52 (int), wk53 (int), wk54
#>   (int), wk55 (int), wk56 (int), wk57 (int), wk58 (int), wk59 (int), wk60
#>   (int), wk61 (int), wk62 (int), wk63 (int), wk64 (int), wk65 (int), wk66
#>   (lgl), wk67 (lgl), wk68 (lgl), wk69 (lgl), wk70 (lgl), wk71 (lgl), wk72
#>   (lgl), wk73 (lgl), wk74 (lgl), wk75 (lgl), wk76 (lgl)
```

Чтобы сделать этот набор данных аккуратным, мы прежде всего должны собрать вместе столбцы `wk`. Имена столбцов зададут переменную `week`, а значения - переменную `rank`:

```
billboard2 <- billboard %>%
  gather(week, rank, wk1:wk76, na.rm = TRUE)
billboard2
#> Source: local data frame [5,307 x 7]
#>
#>     year         artist                    track  time date.entered   week
#>    (int)          (chr)                    (chr) (chr)        (chr) (fctr)
#> 1   2000          2 Pac Baby Don't Cry (Keep...  4:22   2000-02-26    wk1
#> 2   2000        2Ge+her The Hardest Part Of ...  3:15   2000-09-02    wk1
#> 3   2000   3 Doors Down              Kryptonite  3:53   2000-04-08    wk1
#> 4   2000   3 Doors Down                   Loser  4:24   2000-10-21    wk1
#> 5   2000       504 Boyz           Wobble Wobble  3:35   2000-04-15    wk1
#> 6   2000           98^0 Give Me Just One Nig...  3:24   2000-08-19    wk1
#> 7   2000        A*Teens           Dancing Queen  3:44   2000-07-08    wk1
#> 8   2000        Aaliyah           I Don't Wanna  4:15   2000-01-29    wk1
#> 9   2000        Aaliyah               Try Again  4:03   2000-03-18    wk1
#> 10  2000 Adams, Yolanda           Open My Heart  5:30   2000-08-26    wk1
#> ..   ...            ...                     ...   ...          ...    ...
#> Variables not shown: rank (int)
```

Здесь мы использовали `na.rm`, чтобы удалить любые пропущенные значения в объединенном столбце. В этих данных пропущенные значения представляют недели, когда песня не была в чартах, поэтому от них можно безболезненно избавиться.
В данном случае также целесообразно выполнить небольшую очистку, превратив переменную `week` в числовую и вычислив дату, соответствующую каждой неделе в чартах:

```
billboard3 <- billboard2 %>%
  mutate(
    week = extract_numeric(week),
    date = as.Date(date.entered) + 7 * (week - 1)) %>%
  select(-date.entered)
billboard3
#> Source: local data frame [5,307 x 7]
#>
#>     year         artist                    track  time  week  rank
#>    (int)          (chr)                    (chr) (chr) (dbl) (int)
#> 1   2000          2 Pac Baby Don't Cry (Keep...  4:22     1    87
#> 2   2000        2Ge+her The Hardest Part Of ...  3:15     1    91
#> 3   2000   3 Doors Down              Kryptonite  3:53     1    81
#> 4   2000   3 Doors Down                   Loser  4:24     1    76
#> 5   2000       504 Boyz           Wobble Wobble  3:35     1    57
#> 6   2000           98^0 Give Me Just One Nig...  3:24     1    51
#> 7   2000        A*Teens           Dancing Queen  3:44     1    97
#> 8   2000        Aaliyah           I Don't Wanna  4:15     1    84
#> 9   2000        Aaliyah               Try Again  4:03     1    59
#> 10  2000 Adams, Yolanda           Open My Heart  5:30     1    76
#> ..   ...            ...                     ...   ...   ...   ...
#> Variables not shown: date (date)
```

Наконец, никогда не лишним будет отсортировать данные. Мы можем сделать это по исполнителю, песне и неделе:

```
billboard3 %>% arrange(artist, track, week)
#> Source: local data frame [5,307 x 7]
#>
#>     year  artist                  track  time  week  rank       date
#>    (int)   (chr)                  (chr) (chr) (dbl) (int)     (date)
#> 1   2000   2 Pac Baby Don't Cry (Keep...  4:22     1    87 2000-02-26
#> 2   2000   2 Pac Baby Don't Cry (Keep...  4:22     2    82 2000-03-04
#> 3   2000   2 Pac Baby Don't Cry (Keep...  4:22     3    72 2000-03-11
#> 4   2000   2 Pac Baby Don't Cry (Keep...  4:22     4    77 2000-03-18
#> 5   2000   2 Pac Baby Don't Cry (Keep...  4:22     5    87 2000-03-25
#> 6   2000   2 Pac Baby Don't Cry (Keep...  4:22     6    94 2000-04-01
#> 7   2000   2 Pac Baby Don't Cry (Keep...  4:22     7    99 2000-04-08
#> 8   2000 2Ge+her The Hardest Part Of ...  3:15     1    91 2000-09-02
#> 9   2000 2Ge+her The Hardest Part Of ...  3:15     2    87 2000-09-09
#> 10  2000 2Ge+her The Hardest Part Of ...  3:15     3    92 2000-09-16
#> ..   ...     ...                    ...   ...   ...   ...        ...
```

Или по дате и месту в чарте:

```
billboard3 %>% arrange(date, rank)
#> Source: local data frame [5,307 x 7]
#>
#>     year   artist  track  time  week  rank       date
#>    (int)    (chr)  (chr) (chr) (dbl) (int)     (date)
#> 1   2000 Lonestar Amazed  4:25     1    81 1999-06-05
#> 2   2000 Lonestar Amazed  4:25     2    54 1999-06-12
#> 3   2000 Lonestar Amazed  4:25     3    44 1999-06-19
#> 4   2000 Lonestar Amazed  4:25     4    39 1999-06-26
#> 5   2000 Lonestar Amazed  4:25     5    38 1999-07-03
#> 6   2000 Lonestar Amazed  4:25     6    33 1999-07-10
#> 7   2000 Lonestar Amazed  4:25     7    29 1999-07-17
#> 8   2000    Amber Sexual  4:38     1    99 1999-07-17
#> 9   2000 Lonestar Amazed  4:25     8    29 1999-07-24
#> 10  2000    Amber Sexual  4:38     2    99 1999-07-24
#> ..   ...      ...    ...   ...   ...   ...        ...
```

Несколько переменных хранятся в одном столбце

Бывает, что после сборки столбцов столбец со значениями ключей является комбинацией нескольких переменных, лежащих в его основе - как в наборе данных `tb` (туберкулез), показанном ниже. Это набор данных ВОЗ, содержащий количество подтвержденных случаев туберкулеза по странам (`country`), годам (`year`) и демографическим группам. Демографические группы разбиты по полу (`sex` : m, f) и возрасту (`age` : 0-14, 15-25, 25-34, 35-44, 45-54, 55-64, неизвестно).

```
tb <- tbl_df(read.csv("tb.csv", stringsAsFactors = FALSE))
tb
#> Source: local data frame [5,769 x 22]
#>
#>     iso2  year   m04  m514  m014 m1524 m2534 m3544 m4554 m5564   m65    mu
#>    (chr) (int) (int) (int) (int) (int) (int) (int) (int) (int) (int) (int)
#> 1     AD  1989    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 2     AD  1990    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 3     AD  1991    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 4     AD  1992    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 5     AD  1993    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 6     AD  1994    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
#> 7     AD  1996    NA    NA     0     0     0     4     1     0     0    NA
#> 8     AD  1997    NA    NA     0     0     1     2     2     1     6    NA
#> 9     AD  1998    NA    NA     0     0     0     1     0     0     0    NA
#> 10    AD  1999    NA    NA     0     0     0     1     1     0     0    NA
#> ..   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
#> Variables not shown: f04 (int), f514 (int), f014 (int), f1524 (int), f2534
#>   (int), f3544 (int), f4554 (int), f5564 (int), f65 (int), fu (int)
```

Сначала мы соберем столбцы, не являющиеся переменными:

```
tb2 <- tb %>%
  gather(demo, n, -iso2, -year, na.rm = TRUE)
tb2
#> Source: local data frame [35,750 x 4]
#>
#>     iso2  year   demo     n
#>    (chr) (int) (fctr) (int)
#> 1     AD  2005    m04     0
#> 2     AD  2006    m04     0
#> 3     AD  2008    m04     0
#> 4     AE  2006    m04     0
#> 5     AE  2007    m04     0
#> 6     AE  2008    m04     0
#> 7     AG  2007    m04     0
#> 8     AL  2005    m04     0
#> 9     AL  2006    m04     1
#> 10    AL  2007    m04     0
#> ..   ...   ...    ...   ...
```

Заголовки столбцов в таком формате часто содержат в качестве разделителей небуквенные символы (например, `.`, `-`, `_`, `:`) или имеют формат фиксированной ширины, как в этом наборе данных. Функция `separate()` позволяет легко разделить составную переменную на отдельные переменные. Вы можете задать регулярное выражение, по которому происходит разбивка (по умолчанию - по небуквенным символам), или вектор с номерами позиций символов. В данном случае мы хотим выполнить разделение после первого символа:

```
tb3 <- tb2 %>%
  separate(demo, c("sex", "age"), 1)
tb3
#> Source: local data frame [35,750 x 5]
#>
#>     iso2  year   sex   age     n
#>    (chr) (int) (chr) (chr) (int)
#> 1     AD  2005     m    04     0
#> 2     AD  2006     m    04     0
#> 3     AD  2008     m    04     0
#> 4     AE  2006     m    04     0
#> 5     AE  2007     m    04     0
#> 6     AE  2008     m    04     0
#> 7     AG  2007     m    04     0
#> 8     AL  2005     m    04     0
#> 9     AL  2006     m    04     1
#> 10    AL  2007     m    04     0
#> ..   ...   ...   ...   ...   ...
```

Хранение данных в такой форме решает проблему исходных данных. Мы хотим сравнить доли, а не количества, т.е. мы должны знать численность населения. В исходном формате нет простого способа для добавления переменной с численностью населения. Она должна храниться в отдельной таблице, что усложняет корректное сопоставление населения и количества случаев туберкулеза. В аккуратной форме добавление переменных для населения и доли не представляет сложности, потому что это просто дополнительные столбцы.

Переменные хранятся и в строках, и в столбцах

Наиболее сложной формой беспорядочных данных является ситуация, когда переменные хранятся и в строках, и в столбцах. Представленный ниже фрагмент кода загружает данные с одной метеостанции (MX17004). Эти данные из Global Historical Climatology Network содержат ежедневную информацию о погоде в Мексике за пять месяцев 2010 года.

```
weather <- tbl_df(read.csv("weather.csv", stringsAsFactors = FALSE))
weather
#> Source: local data frame [22 x 35]
#>
#>         id  year month element    d1    d2    d3    d4    d5    d6    d7
#>      (chr) (int) (int)   (chr) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
#> 1  MX17004  2010     1    tmax    NA    NA    NA    NA    NA    NA    NA
#> 2  MX17004  2010     1    tmin    NA    NA    NA    NA    NA    NA    NA
#> 3  MX17004  2010     2    tmax    NA  27.3  24.1    NA    NA    NA    NA
#> 4  MX17004  2010     2    tmin    NA  14.4  14.4    NA    NA    NA    NA
#> 5  MX17004  2010     3    tmax    NA    NA    NA    NA  32.1    NA    NA
#> 6  MX17004  2010     3    tmin    NA    NA    NA    NA  14.2    NA    NA
#> 7  MX17004  2010     4    tmax    NA    NA    NA    NA    NA    NA    NA
#> 8  MX17004  2010     4    tmin    NA    NA    NA    NA    NA    NA    NA
#> 9  MX17004  2010     5    tmax    NA    NA    NA    NA    NA    NA    NA
#> 10 MX17004  2010     5    tmin    NA    NA    NA    NA    NA    NA    NA
#> ..     ...   ...   ...     ...   ...   ...   ...   ...   ...   ...   ...
#> Variables not shown: d8 (dbl), 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)
```

Набор данных содержит переменные в отдельных столбцах (`id` , `year` , `month`); переменные, разбросанные по столбцам (`day` , d1-d31); перменные, разбросанные по строкам (`tmin` , `tmax` - минимальная и максимальная температура). Месяцы, в которых меньше 31 дня, содержат структурные пропущенные значения для последнего дня / последних дней.
Чтобы сделать этот набор данных аккуратным, мы сперва соберем столбцы, соответствующие дням:

```
weather2 <- weather %>%
  gather(day, value, d1:d31, na.rm = TRUE)
weather2
#> Source: local data frame [66 x 6]
#>
#>         id  year month element    day value
#>      (chr) (int) (int)   (chr) (fctr) (dbl)
#> 1  MX17004  2010    12    tmax     d1  29.9
#> 2  MX17004  2010    12    tmin     d1  13.8
#> 3  MX17004  2010     2    tmax     d2  27.3
#> 4  MX17004  2010     2    tmin     d2  14.4
#> 5  MX17004  2010    11    tmax     d2  31.3
#> 6  MX17004  2010    11    tmin     d2  16.3
#> 7  MX17004  2010     2    tmax     d3  24.1
#> 8  MX17004  2010     2    tmin     d3  14.4
#> 9  MX17004  2010     7    tmax     d3  28.6
#> 10 MX17004  2010     7    tmin     d3  17.5
#> ..     ...   ...   ...     ...    ...   ...
```

Для наглядности я отбросил пропущенные значения, сделав их неявными вместо явных. Это нормально, поскольку мы знаем, сколько дней в каждом месяце и легко можем восстановить пропущенные значения в явном виде.
Также выполним небольшую очистку:

```
weather3 <- weather2 %>%
  mutate(day = extract_numeric(day)) %>%
  select(id, year, month, day, element, value) %>%
  arrange(id, year, month, day)
weather3
#> Source: local data frame [66 x 6]
#>
#>         id  year month   day element value
#>      (chr) (int) (int) (dbl)   (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     2    tmin  14.4
#> 5  MX17004  2010     2     3    tmax  24.1
#> 6  MX17004  2010     2     3    tmin  14.4
#> 7  MX17004  2010     2    11    tmax  29.7
#> 8  MX17004  2010     2    11    tmin  13.4
#> 9  MX17004  2010     2    23    tmax  29.9
#> 10 MX17004  2010     2    23    tmin  10.7
#> ..     ...   ...   ...   ...     ...   ...
```

Этот набор данных уже почти аккуратный, но столбец `element` не является переменной; он содержит имена переменных. (В этом примере не показаны другие метеорологические переменные - `prcp` (осадки) `snow` (снегопад)). Чтобы это исправить, нужна операция распространения. Она представляет собой действие, обратное сборке - `element` и `value` превращаются в столбцы:

```
weather3 %>% spread(element, value)
#> Source: local data frame [33 x 6]
#>
#>         id  year month   day  tmax  tmin
#>      (chr) (int) (int) (dbl) (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
#> 7  MX17004  2010     3    10  34.5  16.8
#> 8  MX17004  2010     3    16  31.1  17.6
#> 9  MX17004  2010     4    27  36.3  16.7
#> 10 MX17004  2010     5    27  33.2  18.2
#> ..     ...   ...   ...   ...   ...   ...
```

Это аккуратная форма: одна переменная в каждом столбце, каждая строка представляет один день.

Несколько типов единиц наблюдения в одной таблице

Наборы данных часто включают значения, собранные на разных уровнях, для разных типов единиц наблюдения. В процессе приведения к аккуратному виду каждый тип единиц наблюдения должен храниться в отдельной таблице. Такой подход тесно связан с идеей нормализации баз данных, когда каждый факт выражается только в одном месте. Это важно, потому что в противном случае могут возникнуть несоответствия.
Набор данных `billboard` на самом деле содержит наблюдения по двум типам единиц наблюдения: песня и ее место в чарте за каждую неделю. Это проявляется в дублировании фактов о песне: `artist`, `year` и `time` повторяются много раз.
Этот набор данных должен быть разбит на две части: набор данных `song`, содержащий переменные `artist`, `song name` и `time`, и набор данных `rank`, содержащий место в чарте (`rank`) для каждой песни (`song`) за каждую неделю (`week`). Сначала извлечем набор данных `song`:

```
song <- billboard3 %>%
  select(artist, track, year, time) %>%
  unique() %>%
  mutate(song_id = row_number())
song
#> Source: local data frame [317 x 5]
#>
#>            artist                    track  year  time song_id
#>             (chr)                    (chr) (int) (chr)   (int)
#> 1          2 Pac Baby Don't Cry (Keep...  2000  4:22       1
#> 2         2Ge+her The Hardest Part Of ...  2000  3:15       2
#> 3    3 Doors Down              Kryptonite  2000  3:53       3
#> 4    3 Doors Down                   Loser  2000  4:24       4
#> 5        504 Boyz           Wobble Wobble  2000  3:35       5
#> 6           98^0 Give Me Just One Nig...  2000  3:24       6
#> 7         A*Teens           Dancing Queen  2000  3:44       7
#> 8         Aaliyah           I Don't Wanna  2000  4:15       8
#> 9         Aaliyah               Try Again  2000  4:03       9
#> 10 Adams, Yolanda           Open My Heart  2000  5:30      10
#> ..            ...                     ...   ...   ...     ...
```

Затем используем его для создания набора данных `rank` путем замены повторяющихся фактов о песне на указатель (уникальный id песни):

```
rank <- billboard3 %>%
  left_join(song, c("artist", "track", "year", "time")) %>%
  select(song_id, date, week, rank) %>%
  arrange(song_id, date)
rank
#> Source: local data frame [5,307 x 4]
#>
#>    song_id       date  week  rank
#>      (int)     (date) (dbl) (int)
#> 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
#> 7        1 2000-04-08     7    99
#> 8        2 2000-09-02     1    91
#> 9        2 2000-09-09     2    87
#> 10       2 2000-09-16     3    92
#> ..     ...        ...   ...   ...
```

Вы также можете представить себе набор данных `week` , в котором будет записана фоновая информация о неделе, возможно, общее количество проданных песен или подобная “демографическая” информация.
Нормализация полезна для приведения к аккуратному виду и для устранения несоответствий. Тем не менее, есть мало инструментов для анализа данных, которые работают напрямую с реляционными данными, поэтому анализ обычно также требует денормализации или слияния наборов данных обратно в таблицу.

Один тип единиц наблюдения в нескольких таблицах

Также часто бывает, что значения, касающиеся одного типа единиц наблюдения, разбросаны по нескольким таблицам или файлам. Эти таблицы и файлы часто разделены по другой переменной, так что они представляют отдельный год, человека или место. Пока формат отдельных записей остается постоянным, проблему легко решить:

1. Читаем файлы в список таблиц.
2. Для каждой таблицы добавляем новый столбец, содержащий имя исходного файла (часто оно является важной переменной).
3. Объединяем все таблицы в одну.

Пакет plyr позволяет легко это сделать. Следующий код создает вектор имен файлов в каталоге (`data/`), которые соответствуют регулярному выражению (оканчиваются на `.csv`). Далее мы присваиваем каждому элементу вектора имя файла. Мы делаем это, потому что хотим сохранить имена на следующем этапе, гарантировав, что каждая строка итогового файла будет помечена в соответствии с источником данных. Наконец, `ldply()` обрабатывает каждый путь, читая csv-файлы и объединяя результаты в одну таблицу.

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

Как только у вас есть единая таблица, вы можете выполнить дополнительные действия по приданию аккуратного вида, если это нужно. Пример можно найти в https://github.com/hadley/data-baby-names, где 129 годичных таблиц с именами детей от US Social Security Administration объединяются в один файл.
Более сложная ситуация возникает, когда структура наборов данных изменяется с течением времени. Например, наборы данных могут содержать разные переменные, одни и те же переменные с разными именами, разные обозначения пропущенных значений. Это может потребовать приведения к аккуратному виду каждого отдельного файла (или, если повезет, небольших групп файлов) с их последующим объединением. Примером служит https://github.com/hadley/data-fuel-economy, где показано приведение к аккуратному виду данных EPA по экономии топлива для >50000 машин за 1978-2008 гг. Сырые данные доступны онлайн, но каждый год хранится в отдельном файле, причем есть четыре основных формата со многими вариациями, что делает обработку этих данных с целью придания им аккуратного вида значительной проблемой.