Pivoting

2023 年 6 月 18 日

0.1 Introduction

このヴィネットでは、新しい関数 `pivot_longer()` と `pivot_wider()` の使用法を説明します。その目的は `gather()` と `spread()` の使い勝手を向上させ、他のパッケージで見られる最先端の機能を取り入れることにあります。

以前から、`spread()` と `gather()` のデザインには何か根本的な問題があることは明らかでした。多くの人がこの名前を直感的に理解できず、どの方向が広がり、どの方向が集まるかを覚えるのが難しいのです。また、これらの関数の引数を覚えるのも意外と大変なようで、多くの人が(私も含めて！)毎回ドキュメントを参照しなければならないのです。

R でリシェイプを進めている他の R パッケージからヒントを得た重要な新機能が 2 つあります：

- `pivot_longer()` は、Matt Dowle と Arun Srinivasan による data.table パッケージが提供する拡張 `melt()` と `dcast()` 関数に触発されて、異なる型を持つ複数の値変数を扱うことができます。
- `pivot_longer()` と `pivot_wider()` は、John Mount と Nina Zumel による cdata パッケージに触発されて、列名に格納されたメタデータがデータ変数になる(あるいはその逆)方法を正確に指定するデータフレームを受け取ることができます。

このビネットでは、`pivot_longer()` と `pivot_wider()` を使用して、単純なものから複雑なものまで、様々なデータ再形成の課題を解決する様子をご覧いただきながら、そのキーアイデアを学びます。

まず始めに、必要なパッケージをいくつかロードします。実際の解析コードでは、`library(tidyverse)` で行うかと思いますが、このヴィネットはパッケージに埋め込まれているので、ここではそれができないのです。

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

0.2 Longer

`pivot_longer()` は、行数を増やし、列数を減らすことで、データセットを**長く**する。データセットが「長い形」であると表現することに意味があるとは思いません。長さは相対的な用語であり、(例えば)データセット A

はデータセット B より長いとしか言いようがありません。

自然界に存在するデータセットは、分析のしやすさよりもデータ入力のしやすさや比較のしやすさに最適化されていることが多いため、`pivot_longer()` は整頓によく用いられます。以下のセクションでは、現実的な幅広いデータセットに対して `pivot_longer()` を使用する方法を紹介します。

0.2.1 String data in column names

relig_income‘データセットには、宗教と年収を尋ねたアンケートに基づくカウントが格納されています：

```
relig_income
#> # A tibble: 18 x 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 t ~     15        14        15        11        10        35         21
#>  6 Evangel~     575       869      1064       982       881      1486        949
#>  7 Hindu          1         9         7         9        11        34         47
#>  8 Histori~     228       244       236       238       197       223        131
#>  9 Jehovah~      20        27        24        24        21        30         15
#> 10 Jewish        19        19        25        25        30        95         69
#> # i 8 more rows
#> # i 3 more variables: `$100-150k` <dbl>, `>150k` <dbl>,
#> #   `Don't know/refused` <dbl>
```

このデータセットには 3 つの変数が含まれています：

- 行に格納されている `religion`、
- 列の名前に広がる `income` と、
- セルの値に格納される `count` です。

それを整理するために、`pivot_longer()` を使用します：

```
relig_income %>%
  pivot_longer(
    cols = !religion,
    names_to = "income",
    values_to = "count"
  )
#> # A tibble: 180 x 3
```

```
#>    religion income             count
#>    <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
#>  7 Agnostic $75-100k            122
#>  8 Agnostic $100-150k           109
#>  9 Agnostic >150k                84
#> 10 Agnostic Don't know/refused   96
#> # i 170 more rows
```

- 最初の引数は、整形するデータセット relig_income です。
- cols は、どのカラムの形を変える必要があるかを記述します。この場合この場合、religion を除いたすべてのカラムが対象となります。
- names_to は、カラム名に格納されたデータから作成される変数の名前です。income のように、カラム名に格納されたデータから作成される変数の名前を指定します。
- セルの値に格納されているデータから作成される変数の名前、つまり values_to を指定します。セル値に格納されたデータから生成される変数の名前、すなわち count を指定します。

relig_income には names_to と values_to のどちらのカラムも存在しないので、引用符で囲まれた文字列として提供します。

0.2.2 Numeric data in column names

billboard データセットには、2000 年における楽曲のビルボードランキングが記録されている。これは relig_income データと似た形式を持つが、カラム名にエンコードされているデータは文字列ではなく、実際には数値である。

```
billboard
#> # A tibble: 317 x 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~ 2000-02-26      87    82    72    77    87    94    99    NA
#>  2 2Ge+her    The ~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
#>  3 3 Doors D~ Kryp~ 2000-04-08      81    70    68    67    66    57    54    53
#>  4 3 Doors D~ Loser 2000-10-21      76    76    72    69    67    65    55    59
```

```
#>  5 504 Boyz   Wobb~ 2000-04-15      57    34    25    17    17    31    36    49
#>  6 98^0       Give~ 2000-08-19      51    39    34    26    26    19     2     2
#>  7 A*Teens    Danc~ 2000-07-08      97    97    96    95   100    NA    NA    NA
#>  8 Aaliyah    I Do~ 2000-01-29      84    62    51    41    38    35    35    38
#>  9 Aaliyah    Try ~ 2000-03-18      59    53    38    28    21    18    16    14
#> 10 Adams, Yo~ Open~ 2000-08-26      76    76    74    69    68    67    61    58
#> # i 307 more rows
#> # i 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>, ...
```

まず、relig_income データセットと同じ基本仕様で始めることができる。ここでは、名前を week という変数にし、値を rank という変数にしたい。また、values_drop_na を使用して、欠損値に対応する行を削除しています。すべての曲が 76 週すべてチャートにとどまるわけではないので、入力データの構造上、不必要な明示的な NA を作成せざるを得ない。

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  )
#> # A tibble: 5,307 x 5
#>    artist  track                   date.entered week   rank
#>    <chr>   <chr>                   <date>       <chr> <dbl>
#>  1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk1      87
#>  2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk2      82
#>  3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk3      72
#>  4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk4      77
#>  5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk5      87
#>  6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk6      94
#>  7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk7      99
#>  8 2Ge+her The Hardest Part Of ... 2000-09-02   wk1      91
#>  9 2Ge+her The Hardest Part Of ... 2000-09-02   wk2      87
#> 10 2Ge+her The Hardest Part Of ... 2000-09-02   wk3      92
```

```
#> # i 5,297 more rows
```

各曲がどれくらいの期間チャートに入っていたかを簡単に調べることができればいいのですが、そのためには week 変数を整数に変換することが必要でしょう。names_prefix は wk という接頭辞を取り除き、names_transform は week を整数に変換します：

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    names_prefix = "wk",
    names_transform = as.integer,
    values_to = "rank",
    values_drop_na = TRUE,
  )
```

また、数値以外の成分を自動的に除去する readr::parse_number() を使えば、1 つの引数でこれを行うことができます：

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    names_transform = readr::parse_number,
    values_to = "rank",
    values_drop_na = TRUE,
  )
```

0.2.3 Many variables in column names

より困難な状況は、カラム名に複数の変数が詰め込まれている場合に発生します。例えば、who データセットを見てみましょう：

```
who
#> # A tibble: 7,240 x 60
#>    country  iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534 new_sp_m3544
#>    <chr>    <chr> <chr> <dbl>       <dbl>        <dbl>        <dbl>        <dbl>
#>  1 Afghani~ AF    AFG    1980          NA           NA           NA           NA
#>  2 Afghani~ AF    AFG    1981          NA           NA           NA           NA
#>  3 Afghani~ AF    AFG    1982          NA           NA           NA           NA
```

```
#>  4 Afghani~ AF    AFG    1983          NA          NA          NA          NA
#>  5 Afghani~ AF    AFG    1984          NA          NA          NA          NA
#>  6 Afghani~ AF    AFG    1985          NA          NA          NA          NA
#>  7 Afghani~ AF    AFG    1986          NA          NA          NA          NA
#>  8 Afghani~ AF    AFG    1987          NA          NA          NA          NA
#>  9 Afghani~ AF    AFG    1988          NA          NA          NA          NA
#> 10 Afghani~ AF    AFG    1989          NA          NA          NA          NA
#> # i 7,230 more rows
#> # i 52 more variables: new_sp_m4554 <dbl>, new_sp_m5564 <dbl>,
#> #   new_sp_m65 <dbl>, new_sp_f014 <dbl>, new_sp_f1524 <dbl>,
#> #   new_sp_f2534 <dbl>, new_sp_f3544 <dbl>, new_sp_f4554 <dbl>,
#> #   new_sp_f5564 <dbl>, new_sp_f65 <dbl>, new_sn_m014 <dbl>,
#> #   new_sn_m1524 <dbl>, new_sn_m2534 <dbl>, new_sn_m3544 <dbl>,
#> #   new_sn_m4554 <dbl>, new_sn_m5564 <dbl>, new_sn_m65 <dbl>, ...
```

country、iso2、iso3、year はすでに変数なので、そのままでよい。しかし、new_sp_m014 から newrel_f65 までのカラムは、その名前に 4 つの変数をエンコードしています：

- 接頭辞の new_/new は、これらが新規症例のカウントであることを示す。このデータセットには新規のケースしか含まれていないので、ここでは一定なので無視することにする。
- sp/rel/ep は、症例がどのように診断されたかを説明する。
- m/f は性別を表します。
- 014/1524/2535/3544/4554/65 は、年齢層を表します。

names_to で複数のカラム名を指定し、names_sep または names_pattern を指定することで、これらの変数を分割することができます。ここでは names_pattern が最も自然にフィットします。これは extract と似たようなインターフェイスを持っています。グループを含む正規表現 (() で定義) を与えると、各グループをカラムに配置します。

```
who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = c("diagnosis", "gender", "age"),
    names_pattern = "new_?(.*)_(.)(.*)",
    values_to = "count"
  )
#> # A tibble: 405,440 x 8
#>    country     iso2  iso3   year diagnosis gender age   count
#>    <chr>       <chr> <chr> <dbl> <chr>     <chr>  <chr> <dbl>
#>  1 Afghanistan AF    AFG    1980 sp        m      014      NA
```

```
#>  2 Afghanistan AF    AFG    1980 sp        m      1524     NA
#>  3 Afghanistan AF    AFG    1980 sp        m      2534     NA
#>  4 Afghanistan AF    AFG    1980 sp        m      3544     NA
#>  5 Afghanistan AF    AFG    1980 sp        m      4554     NA
#>  6 Afghanistan AF    AFG    1980 sp        m      5564     NA
#>  7 Afghanistan AF    AFG    1980 sp        m      65       NA
#>  8 Afghanistan AF    AFG    1980 sp        f      014      NA
#>  9 Afghanistan AF    AFG    1980 sp        f      1524     NA
#> 10 Afghanistan AF    AFG    1980 sp        f      2534     NA
#> # i 405,430 more rows
```

さらに一歩進んで、readr 関数を使用して性別と年齢を因数に変換することができます。これは、既知の値のセットを持つカテゴリ変数がある場合、良い練習になると思います。

```
who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = c("diagnosis", "gender", "age"),
    names_pattern = "new_?(.*)_(.)(.*)",
    names_transform = list(
      gender = ~ readr::parse_factor(.x, levels = c("f", "m")),
      age = ~ readr::parse_factor(
        .x,
        levels = c("014", "1524", "2534", "3544", "4554", "5564", "65"),
        ordered = TRUE
      )
    ),
    values_to = "count",
)
```

この方法で実行すると、事後的に変更を行うよりも少し効率的です。`mutate()` では多くの繰り返しを変換する必要があるのに対し、`pivot_longer()` は各名前の 1 つの出現を変換するだけで済みます。

0.2.4 Multiple observations per row

これまで、1 行に 1 つの観測値を持つデータフレームを扱ってきましたが、重要なピボット問題の多くは、1 行に複数の観測値を持ちます。この場合、出力に表示したい列の名前が、入力の列名の一部であることから、通常はこのケースを認識できます。このセクションでは、このようなデータをピボットする方法について学びます。

以下の例は、この問題に対する tidyr の解決策のインスピレーションとして、data.table vignette から引用しています。

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

各子供について name と dob (date of birth) という 2 つの情報 (または値) を持っていることに注意してください。これらは最終的に別のカラムに格納する必要があります。ここでも names_to に複数の変数を指定し、names_sep を使って各変数名を分割しています。これは pivot_longer() に、列名の一部が測定される「値」(出力では変数となる)を指定するものであることを伝えるものです。

```
household %>%
  pivot_longer(
    cols = !family,
    names_to = c(".value", "child"),
    names_sep = "_",
    values_drop_na = TRUE
  )
#> # A tibble: 9 x 4
#>   family child  dob        name
#>    <int> <chr>  <date>     <chr>
#> 1      1 child1 1998-11-26 Susan
#> 2      1 child2 2000-01-29 Jose
#> 3      2 child1 1996-06-22 Mark
#> 4      3 child1 2002-07-11 Sam
#> 5      3 child2 2004-04-05 Seth
#> 6      4 child1 2004-10-10 Craig
#> 7      4 child2 2009-08-27 Khai
#> 8      5 child1 2000-12-05 Parker
#> 9      5 child2 2005-02-28 Gracie
```

values_drop_na = TRUE‘ の使用に注意してください：入力形状は、存在しないオブザベーションのための明示的な欠損変数を作成することを強制します。

同様の問題は、ベース R に組み込まれている anscombe データセットにも存在します：

```
anscombe
#>    x1 x2 x3 x4    y1   y2    y3    y4
#> 1  10 10 10  8  8.04 9.14  7.46  6.58
#> 2   8  8  8  8  6.95 8.14  6.77  5.76
#> 3  13 13 13  8  7.58 8.74 12.74  7.71
#> 4   9  9  9  8  8.81 8.77  7.11  8.84
#> 5  11 11 11  8  8.33 9.26  7.81  8.47
#> 6  14 14 14  8  9.96 8.10  8.84  7.04
#> 7   6  6  6  8  7.24 6.13  6.08  5.25
#> 8   4  4  4 19  4.26 3.10  5.39 12.50
#> 9  12 12 12  8 10.84 9.13  8.15  5.56
#> 10  7  7  7  8  4.82 7.26  6.42  7.91
#> 11  5  5  5  8  5.68 4.74  5.73  6.89
```

このデータセットには 4 組の変数(x1 と y1，x2 と y2 など)が含まれており，Anscombe のカルテット(平均、標準偏差など、要約統計量は同じだが, データが全く異なる 4 つのデータセットを集めたもの)の根底にある. 私たちは、列 set、x、y を持つデータセットを作りたいと考えています。

```
anscombe %>%
  pivot_longer(
    cols = everything(),
    cols_vary = "slowest",
    names_to = c(".value", "set"),
    names_pattern = "(.)(.)"
  )
#> # A tibble: 44 x 3
#>    set       x     y
#>    <chr> <dbl> <dbl>
#>  1 1        10  8.04
#>  2 1         8  6.95
#>  3 1        13  7.58
#>  4 1         9  8.81
#>  5 1        11  8.33
#>  6 1        14  9.96
#>  7 1         6  7.24
#>  8 1         4  4.26
#>  9 1        12 10.8
#> 10 1         7  4.82
#> # i 34 more rows
```

`cols_vary` を `"slowest"` に設定すると、`x1` と `y1` の列の値をまとめて出力し、その後 `x2` と `y2` に移行します。この引数は、データセットのすべての列をピボット処理する場合に、より直感的に順序付けられた出力を生成することがよくあります。

パネルデータでも同様の事態が起こりうる。例えば、Thomas Leeper が提供したこのデータセットの例を見てみましょう。`anscombe` と同じ手法で整頓することができます：

```
pnl <- tibble(
  x = 1:4,
  a = c(1, 1,0, 0),
  b = c(0, 1, 1, 1),
  y1 = rnorm(4),
  y2 = rnorm(4),
  z1 = rep(3, 4),
  z2 = rep(-2, 4),
)

pnl %>%
  pivot_longer(
    cols = !c(x, a, b),
    names_to = c(".value", "time"),
    names_pattern = "(.)(.)"
  )
#> # A tibble: 8 x 6
#>       x     a     b time       y     z
#>   <int> <dbl> <dbl> <chr>  <dbl> <dbl>
#> 1     1     1     0 1     -0.443     3
#> 2     1     1     0 2      0.862    -2
#> 3     2     1     1 1      0.280     3
#> 4     2     1     1 2     -0.401    -2
#> 5     3     0     1 1      1.08      3
#> 6     3     0     1 2      0.501    -2
#> 7     4     0     1 1     -0.782     3
#> 8     4     0     1 2      0.575    -2
```

0.3 Wider

`pivot_wider()` は、`pivot_longer()` の逆で、列数を増やし、行数を減らすことでデータセットを**広く**するものです。整頓されたデータを作るために `pivot_wider()` が必要になることは比較的少ないですが、プレゼンテーション用の要約表や、他のツールで用いられる形式のデータを作成する際にはよく役に立ちます。

0.3.1 Capture-recapture data

Myfanwy Johnston が提供した `fish_encounters` データセットは、川を泳ぐ魚が自動監視局によって検出されたときの様子を記述しています：

```
fish_encounters
#> # A tibble: 114 x 3
#>    fish  station  seen
#>    <fct> <fct>   <int>
#>  1 4842  Release     1
#>  2 4842  I80_1       1
#>  3 4842  Lisbon      1
#>  4 4842  Rstr        1
#>  5 4842  Base_TD     1
#>  6 4842  BCE         1
#>  7 4842  BCW         1
#>  8 4842  BCE2        1
#>  9 4842  BCW2        1
#> 10 4842  MAE         1
#> # i 104 more rows
```

このデータを分析するために用いられる多くのツールは、各ステーションが列となる形でデータを必要とします：

```
fish_encounters %>%
  pivot_wider(
    names_from = station,
    values_from = seen
  )
#> # A tibble: 19 x 12
#>    fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MAW
#>    <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int>
#>  1 4842        1     1      1     1       1     1     1     1     1     1     1
#>  2 4843        1     1      1     1       1     1     1     1     1     1     1
#>  3 4844        1     1      1     1       1     1     1     1     1     1     1
#>  4 4845        1     1      1     1       1    NA    NA    NA    NA    NA    NA
#>  5 4847        1     1      1    NA      NA    NA    NA    NA    NA    NA    NA
#>  6 4848        1     1      1     1      NA    NA    NA    NA    NA    NA    NA
#>  7 4849        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
#>  8 4850        1     1     NA     1       1     1     1    NA    NA    NA    NA
```

```
#>  9 4851        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
#> 10 4854        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
#> # i 9 more rows
```

このデータセットでは、魚がステーションで検出された場合のみ記録され、検出されなかった場合は記録されません（これはこの種のデータではよくあることです）。つまり、出力データは NA で埋め尽くされます。しかし、この場合、記録がないということは魚が「目撃（seen）されなかった」ということなので、pivot_wider() に欠損値をゼロで埋めてもらうことができます：

```
fish_encounters %>%
  pivot_wider(
    names_from = station,
    values_from = seen,
    values_fill = 0
  )
#> # A tibble: 19 x 12
#>    fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MAW
#>    <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int>
#>  1 4842        1     1      1     1       1     1     1     1     1     1     1
#>  2 4843        1     1      1     1       1     1     1     1     1     1     1
#>  3 4844        1     1      1     1       1     1     1     1     1     1     1
#>  4 4845        1     1      1     1       1     0     0     0     0     0     0
#>  5 4847        1     1      1     0       0     0     0     0     0     0     0
#>  6 4848        1     1      1     1       0     0     0     0     0     0     0
#>  7 4849        1     1      0     0       0     0     0     0     0     0     0
#>  8 4850        1     1      0     1       1     1     1     0     0     0     0
#>  9 4851        1     1      0     0       0     0     0     0     0     0     0
#> 10 4854        1     1      0     0       0     0     0     0     0     0     0
#> # i 9 more rows
```

0.3.2 Aggregation

また、pivot_wider() を使って単純な集計を行うこともできます。例えば、ベース R に組み込まれている warpbreaks データセットを見てみましょう（より良いプリント手法のために tibble に変換されています）：

```
warpbreaks <- warpbreaks %>%
  as_tibble() %>%
  select(wool, tension, breaks)
warpbreaks
#> # A tibble: 54 x 3
```

```
#>    wool  tension breaks
#>    <fct> <fct>    <dbl>
#>  1 A     L           26
#>  2 A     L           30
#>  3 A     L           54
#>  4 A     L           25
#>  5 A     L           70
#>  6 A     L           52
#>  7 A     L           51
#>  8 A     L           26
#>  9 A     L           67
#> 10 A     M           18
#> # i 44 more rows
```

これは、wool（A、B）と tension（L、M、H）の組み合わせごとに 9 つの複製を持つ設計実験である：

```
warpbreaks %>%
  count(wool, tension)
#> # A tibble: 6 x 3
#>   wool  tension     n
#>   <fct> <fct>   <int>
#> 1 A     L           9
#> 2 A     M           9
#> 3 A     H           9
#> 4 B     L           9
#> 5 B     M           9
#> 6 B     H           9
```

wool の水準を列にピボットさせようとするとどうなるでしょうか？

```
warpbreaks %>%
  pivot_wider(
    names_from = wool,
    values_from = breaks
  )
#> # A tibble: 3 x 3
#>   tension A         B
#>   <fct>   <list>    <list>
#> 1 L       <dbl [9]> <dbl [9]>
#> 2 M       <dbl [9]> <dbl [9]>
```

```
#> 3 H       <dbl [9]> <dbl [9]>
```

出力の各セルが、入力の複数のセルに対応しているという警告が表示されます。デフォルトの動作では、個々の値をすべて含むリストカラムが生成されます。より有用な出力は、例えば、ウールとテンションの組み合わせごとの平均値などの要約統計です：

```
warpbreaks %>%
  pivot_wider(
    names_from = wool,
    values_from = breaks,
    values_fn = mean
  )
#> # A tibble: 3 x 3
#>   tension     A     B
#>   <fct>   <dbl> <dbl>
#> 1 L        44.6  28.2
#> 2 M        24    28.8
#> 3 H        24.6  18.8
```

より複雑な要約操作の場合は、再形成の前に要約することをお勧めしますが、単純なケースでは、`pivot_wider()` 内で要約するのが便利なことが多いです。

0.3.3 Generate column name from multiple variables

https://stackoverflow.com/questions/24929954 のように、製品、国、年の組み合わせを含む情報があると想像してください。整頓された形では、次のようになります：

```
production <-
  expand_grid(
    product = c("A", "B"),
    country = c("AI", "EI"),
    year = 2000:2014
  ) %>%
  filter((product == "A" & country == "AI") | product == "B") %>%
  mutate(production = rnorm(nrow(.)))
production
#> # A tibble: 45 x 4
#>    product country  year production
#>    <chr>   <chr>   <int>      <dbl>
#>  1 A       AI       2000    -0.922
#>  2 A       AI       2001     0.407
```

```
#>  3 A       AI       2002     2.23
#>  4 A       AI       2003     1.51
#>  5 A       AI       2004     2.77
#>  6 A       AI       2005     1.84
#>  7 A       AI       2006    -0.0340
#>  8 A       AI       2007    -0.203
#>  9 A       AI       2008    -1.46
#> 10 A       AI       2009    -0.146
#> # i 35 more rows
```

データを広げて、product と country の組み合わせごとに 1 つのカラムを持つようにしたいです。このとき、names_from に複数の変数を指定することがポイントです：

```
production %>%
  pivot_wider(
    names_from = c(product, country),
    values_from = production
  )
#> # A tibble: 15 x 4
#>     year    A_AI    B_AI    B_EI
#>    <int>   <dbl>   <dbl>   <dbl>
#>  1  2000 -0.922   0.425   0.324
#>  2  2001  0.407   1.47    0.854
#>  3  2002  2.23    1.01    1.95
#>  4  2003  1.51   -0.658  -2.16
#>  5  2004  2.77    0.613  -1.38
#>  6  2005  1.84    0.403  -0.169
#>  7  2006 -0.0340 -1.20    0.377
#>  8  2007 -0.203   0.0362 -0.0955
#>  9  2008 -1.46   -0.263  -0.451
#> 10  2009 -0.146   2.22    0.214
#> # i 5 more rows
```

names_from または values_from が複数の変数を選択する場合、names_sep や names_prefix 、または names_glue を使って、出力する列名の構成方法を制御することができます：

```
production %>%
  pivot_wider(
    names_from = c(product, country),
    values_from = production,
```

```
    names_sep = ".",
    names_prefix = "prod."
  )
#> # A tibble: 15 x 4
#>     year prod.A.AI prod.B.AI prod.B.EI
#>    <int>     <dbl>     <dbl>     <dbl>
#>  1  2000   -0.922     0.425     0.324
#>  2  2001    0.407     1.47      0.854
#>  3  2002    2.23      1.01      1.95
#>  4  2003    1.51     -0.658    -2.16
#>  5  2004    2.77      0.613    -1.38
#>  6  2005    1.84      0.403    -0.169
#>  7  2006   -0.0340   -1.20      0.377
#>  8  2007   -0.203     0.0362   -0.0955
#>  9  2008   -1.46     -0.263    -0.451
#> 10  2009   -0.146     2.22      0.214
#> # i 5 more rows

production %>%
  pivot_wider(
    names_from = c(product, country),
    values_from = production,
    names_glue = "prod_{product}_{country}"
  )
#> # A tibble: 15 x 4
#>     year prod_A_AI prod_B_AI prod_B_EI
#>    <int>     <dbl>     <dbl>     <dbl>
#>  1  2000   -0.922     0.425     0.324
#>  2  2001    0.407     1.47      0.854
#>  3  2002    2.23      1.01      1.95
#>  4  2003    1.51     -0.658    -2.16
#>  5  2004    2.77      0.613    -1.38
#>  6  2005    1.84      0.403    -0.169
#>  7  2006   -0.0340   -1.20      0.377
#>  8  2007   -0.203     0.0362   -0.0955
#>  9  2008   -1.46     -0.263    -0.451
#> 10  2009   -0.146     2.22      0.214
#> # i 5 more rows
```

0.3.4 Tidy census

`us_rent_income` データセットには、2017 年のアメリカの各州の所得と家賃の中央値に関する情報が含まれています(American Community Survey より、tidycensus パッケージで取得)。

```
us_rent_income
#> # A tibble: 104 x 5
#>    GEOID NAME       variable estimate   moe
#>    <chr> <chr>      <chr>       <dbl> <dbl>
#>  1 01    Alabama    income      24476   136
#>  2 01    Alabama    rent          747     3
#>  3 02    Alaska     income      32940   508
#>  4 02    Alaska     rent         1200    13
#>  5 04    Arizona    income      27517   148
#>  6 04    Arizona    rent          972     4
#>  7 05    Arkansas   income      23789   165
#>  8 05    Arkansas   rent          709     5
#>  9 06    California income      29454   109
#> 10 06    California rent         1358     3
#> # i 94 more rows
```

ここで `estimate` と `moe` はどちらも値のカラムなので、`values_from` に渡すことができます：

```
us_rent_income %>%
  pivot_wider(
    names_from = variable,
    values_from = c(estimate, moe)
  )
#> # A tibble: 52 x 6
#>    GEOID NAME                 estimate_income estimate_rent moe_income moe_rent
#>    <chr> <chr>                          <dbl>         <dbl>      <dbl>    <dbl>
#>  1 01    Alabama                        24476           747        136        3
#>  2 02    Alaska                         32940          1200        508       13
#>  3 04    Arizona                        27517           972        148        4
#>  4 05    Arkansas                       23789           709        165        5
#>  5 06    California                     29454          1358        109        3
#>  6 08    Colorado                       32401          1125        109        5
#>  7 09    Connecticut                    35326          1123        195        5
#>  8 10    Delaware                       31560          1076        247       10
#>  9 11    District of Columbia           43198          1424        681       17
```

```
#> 10 12    Florida                    25952          1077          70         3
#> # i 42 more rows
```

なお、出力列には、変数名が自動的に付加されます。

0.3.5 Implicit missing values

時折、名前変数が因子として符号化されているデータに出会うことがありますが、すべてのデータが表現されるわけではありません。

```
weekdays <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")

daily <- tibble(
  day = factor(c("Tue", "Thu", "Fri", "Mon"), levels = weekdays),
  value = c(2, 3, 1, 5)
)

daily
#> # A tibble: 4 x 2
#>   day   value
#>   <fct> <dbl>
#> 1 Tue       2
#> 2 Thu       3
#> 3 Fri       1
#> 4 Mon       5
```

`pivot_wider()` のデフォルトでは、実際にデータで表現されている値から列を生成しますが、将来的にデータが変更された場合に備えて、可能性のある各レベルの列を含めるとよいかもしれません。

```
daily %>%
  pivot_wider(
    names_from = day,
    values_from = value
  )
#> # A tibble: 1 x 4
#>     Tue   Thu   Fri   Mon
#>   <dbl> <dbl> <dbl> <dbl>
#> 1     2     3     1     5
```

`names_expand` 引数は、暗黙の因子レベルを明示的なものに変え、結果で表現することを強制します。また、レベルの順序を使用してカラム名をソートするので、この場合、より直感的な結果が得られます。

```r
daily %>%
  pivot_wider(
    names_from = day,
    values_from = value,
    names_expand = TRUE
  )
#> # A tibble: 1 x 7
#>     Mon   Tue   Wed   Thu   Fri   Sat   Sun
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1     5     2    NA     3     1    NA    NA
```

If multiple `names_from` columns are provided, `names_expand` will generate a Cartesian product of all possible combinations of the `names_from` values. Notice that the following data has omitted some rows where the percentage value would be 0. `names_expand` allows us to make those explicit during the pivot.

複数の `names_from` カラムを指定すると、`names_expand` は `names_from` 値のすべての可能な組み合わせのデカルト積を生成します。以下のデータでは、パーセント値が 0 となる行がいくつか省略されていることに注意してください。names_expand'を使用すると、ピボットの際にこれらを明示することができます。

```r
percentages <- tibble(
  year = c(2018, 2019, 2020, 2020),
  type = factor(c("A", "B", "A", "B"), levels = c("A", "B")),
  percentage = c(100, 100, 40, 60)
)

percentages
#> # A tibble: 4 x 3
#>    year type  percentage
#>   <dbl> <fct>      <dbl>
#> 1  2018 A            100
#> 2  2019 B            100
#> 3  2020 A             40
#> 4  2020 B             60

percentages %>%
  pivot_wider(
    names_from = c(year, type),
    values_from = percentage,
    names_expand = TRUE,
    values_fill = 0
```

```
  )
#> # A tibble: 1 x 6
#>   `2018_A` `2018_B` `2019_A` `2019_B` `2020_A` `2020_B`
#>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
#> 1      100        0        0      100       40       60
```

A related problem can occur when there are implicit missing factor levels or combinations in the `id_cols`. In this case, there are missing rows (rather than columns) that you'd like to explicitly represent. For this example, we'll modify our `daily` data with a `type` column, and pivot on that instead, keeping `day` as an id column.

```
daily <- mutate(daily, type = factor(c("A", "B", "B", "A")))
daily
#> # A tibble: 4 x 3
#>   day   value type
#>   <fct> <dbl> <fct>
#> 1 Tue       2 A
#> 2 Thu       3 B
#> 3 Fri       1 B
#> 4 Mon       5 A
```

All of our `type` levels are represented in the columns, but we are missing some rows related to the unrepresented `day` factor levels.

```
daily %>%
  pivot_wider(
    names_from = type,
    values_from = value,
    values_fill = 0
  )
#> # A tibble: 4 x 3
#>   day       A     B
#>   <fct> <dbl> <dbl>
#> 1 Tue       2     0
#> 2 Thu       0     3
#> 3 Fri       0     1
#> 4 Mon       5     0
```

We can use `id_expand` in the same way that we used `names_expand`, which will expand out (and sort) the implicit missing rows in the `id_cols`.

```
daily %>%
  pivot_wider(
    names_from = type,
    values_from = value,
    values_fill = 0,
    id_expand = TRUE
  )
#> # A tibble: 7 x 3
#>   day       A     B
#>   <fct> <dbl> <dbl>
#> 1 Mon       5     0
#> 2 Tue       2     0
#> 3 Wed       0     0
#> 4 Thu       0     3
#> 5 Fri       0     1
#> 6 Sat       0     0
#> 7 Sun       0     0
```

0.3.6 Unused columns

Imagine you've found yourself in a situation where you have columns in your data that are completely unrelated to the pivoting process, but you'd still like to retain their information somehow. For example, in `updates` we'd like to pivot on the `system` column to create one row summaries of each county's system updates.

```
updates <- tibble(
  county = c("Wake", "Wake", "Wake", "Guilford", "Guilford"),
  date = c(as.Date("2020-01-01") + 0:2, as.Date("2020-01-03") + 0:1),
  system = c("A", "B", "C", "A", "C"),
  value = c(3.2, 4, 5.5, 2, 1.2)
)

updates
#> # A tibble: 5 x 4
#>   county   date       system value
#>   <chr>    <date>     <chr>  <dbl>
#> 1 Wake     2020-01-01 A        3.2
#> 2 Wake     2020-01-02 B        4
#> 3 Wake     2020-01-03 C        5.5
```

```
#> 4 Guilford 2020-01-03 A        2
#> 5 Guilford 2020-01-04 C        1.2
```

We could do that with a typical `pivot_wider()` call, but we completely lose all information about the `date` column.

```
updates %>%
  pivot_wider(
    id_cols = county,
    names_from = system,
    values_from = value
  )
#> # A tibble: 2 x 4
#>   county       A     B     C
#>   <chr>    <dbl> <dbl> <dbl>
#> 1 Wake       3.2     4   5.5
#> 2 Guilford   2      NA   1.2
```

For this example, we'd like to retain the most recent update date across all systems in a particular county. To accomplish that we can use the `unused_fn` argument, which allows us to summarize values from the columns not utilized in the pivoting process.

```
updates %>%
  pivot_wider(
    id_cols = county,
    names_from = system,
    values_from = value,
    unused_fn = list(date = max)
  )
#> # A tibble: 2 x 5
#>   county       A     B     C date
#>   <chr>    <dbl> <dbl> <dbl> <date>
#> 1 Wake       3.2     4   5.5 2020-01-03
#> 2 Guilford   2      NA   1.2 2020-01-04
```

You can also retain the data but delay the aggregation entirely by using `list()` as the summary function.

```
updates %>%
  pivot_wider(
    id_cols = county,
```

```
    names_from = system,
    values_from = value,
    unused_fn = list(date = list)
  )
#> # A tibble: 2 x 5
#>   county       A     B     C date
#>   <chr>    <dbl> <dbl> <dbl> <list>
#> 1 Wake       3.2     4   5.5 <date [3]>
#> 2 Guilford   2      NA   1.2 <date [2]>
```

0.3.7 Contact list

A final challenge is inspired by Jiena Gu. Imagine you have a contact list that you've copied and pasted from a website:

```
contacts <- tribble(
  ~field, ~value,
  "name", "Jiena McLellan",
  "company", "Toyota",
  "name", "John Smith",
  "company", "google",
  "email", "john@google.com",
  "name", "Huxley Ratcliffe"
)
```

This is challenging because there's no variable that identifies which observations belong together. We can fix this by noting that every contact starts with a name, so we can create a unique id by counting every time we see "name" as the `field`:

```
contacts <- contacts %>%
  mutate(
    person_id = cumsum(field == "name")
  )
contacts
#> # A tibble: 6 x 3
#>   field   value            person_id
#>   <chr>   <chr>                <int>
#> 1 name    Jiena McLellan           1
#> 2 company Toyota                   1
#> 3 name    John Smith               2
```

```
#> 4 company google                  2
#> 5 email   john@google.com         2
#> 6 name    Huxley Ratcliffe        3
```

Now that we have a unique identifier for each person, we can pivot `field` and `value` into the columns:

```
contacts %>%
  pivot_wider(
    names_from = field,
    values_from = value
  )
#> # A tibble: 3 x 4
#>   person_id name             company email
#>       <int> <chr>            <chr>   <chr>
#> 1         1 Jiena McLellan   Toyota  <NA>
#> 2         2 John Smith       google  john@google.com
#> 3         3 Huxley Ratcliffe <NA>    <NA>
```

0.4 Longer, then wider

Some problems can't be solved by pivoting in a single direction. The examples in this section show how you might combine `pivot_longer()` and `pivot_wider()` to solve more complex problems.

0.4.1 World bank

`world_bank_pop` contains data from the World Bank about population per country from 2000 to 2018.

```
world_bank_pop
#> # A tibble: 1,064 x 20
#>    country indicator     `2000`  `2001`  `2002`  `2003`  `2004`  `2005`  `2006`
#>    <chr>   <chr>          <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#>  1 ABW     SP.URB.TOTL    4.16e4 4.20e+4 4.22e+4 4.23e+4 4.23e+4 4.24e+4 4.26e+4
#>  2 ABW     SP.URB.GROW    1.66e0 9.56e-1 4.01e-1 1.97e-1 9.46e-2 1.94e-1 3.67e-1
#>  3 ABW     SP.POP.TOTL    8.91e4 9.07e+4 9.18e+4 9.27e+4 9.35e+4 9.45e+4 9.56e+4
#>  4 ABW     SP.POP.GROW    2.54e0 1.77e+0 1.19e+0 9.97e-1 9.01e-1 1.00e+0 1.18e+0
#>  5 AFE     SP.URB.TOTL    1.16e8 1.20e+8 1.24e+8 1.29e+8 1.34e+8 1.39e+8 1.44e+8
#>  6 AFE     SP.URB.GROW    3.60e0 3.66e+0 3.72e+0 3.71e+0 3.74e+0 3.81e+0 3.81e+0
#>  7 AFE     SP.POP.TOTL    4.02e8 4.12e+8 4.23e+8 4.34e+8 4.45e+8 4.57e+8 4.70e+8
#>  8 AFE     SP.POP.GROW    2.58e0 2.59e+0 2.61e+0 2.62e+0 2.64e+0 2.67e+0 2.70e+0
#>  9 AFG     SP.URB.TOTL    4.31e6 4.36e+6 4.67e+6 5.06e+6 5.30e+6 5.54e+6 5.83e+6
```

```
#> 10 AFG     SP.URB.GROW    1.86e0 1.15e+0 6.86e+0 7.95e+0 4.59e+0 4.47e+0 5.03e+0
#> # i 1,054 more rows
#> # i 11 more variables: `2007` <dbl>, `2008` <dbl>, `2009` <dbl>, `2010` <dbl>,
#> #   `2011` <dbl>, `2012` <dbl>, `2013` <dbl>, `2014` <dbl>, `2015` <dbl>,
#> #   `2016` <dbl>, `2017` <dbl>
```

My goal is to produce a tidy dataset where each variable is in a column. It's not obvious exactly what steps are needed yet, but I'll start with the most obvious problem: year is spread across multiple columns.

```
pop2 <- world_bank_pop %>%
  pivot_longer(
    cols = `2000`:`2017`,
    names_to = "year",
    values_to = "value"
  )
pop2
#> # A tibble: 19,152 x 4
#>    country indicator   year  value
#>    <chr>   <chr>       <chr> <dbl>
#>  1 ABW     SP.URB.TOTL 2000  41625
#>  2 ABW     SP.URB.TOTL 2001  42025
#>  3 ABW     SP.URB.TOTL 2002  42194
#>  4 ABW     SP.URB.TOTL 2003  42277
#>  5 ABW     SP.URB.TOTL 2004  42317
#>  6 ABW     SP.URB.TOTL 2005  42399
#>  7 ABW     SP.URB.TOTL 2006  42555
#>  8 ABW     SP.URB.TOTL 2007  42729
#>  9 ABW     SP.URB.TOTL 2008  42906
#> 10 ABW     SP.URB.TOTL 2009  43079
#> # i 19,142 more rows
```

Next we need to consider the `indicator` variable:

```
pop2 %>%
  count(indicator)
#> # A tibble: 4 x 2
#>   indicator       n
#>   <chr>       <int>
#> 1 SP.POP.GROW  4788
#> 2 SP.POP.TOTL  4788
```

```
#> 3 SP.URB.GROW  4788
#> 4 SP.URB.TOTL  4788
```

Here `SP.POP.GROW` is population growth, `SP.POP.TOTL` is total population, and `SP.URB.*` are the same but only for urban areas. Let's split this up into two variables: `area` (total or urban) and the actual variable (population or growth):

```
pop3 <- pop2 %>%
  separate(indicator, c(NA, "area", "variable"))
pop3
#> # A tibble: 19,152 x 5
#>    country area  variable year  value
#>    <chr>   <chr> <chr>    <chr> <dbl>
#>  1 ABW     URB   TOTL     2000  41625
#>  2 ABW     URB   TOTL     2001  42025
#>  3 ABW     URB   TOTL     2002  42194
#>  4 ABW     URB   TOTL     2003  42277
#>  5 ABW     URB   TOTL     2004  42317
#>  6 ABW     URB   TOTL     2005  42399
#>  7 ABW     URB   TOTL     2006  42555
#>  8 ABW     URB   TOTL     2007  42729
#>  9 ABW     URB   TOTL     2008  42906
#> 10 ABW     URB   TOTL     2009  43079
#> # i 19,142 more rows
```

Now we can complete the tidying by pivoting `variable` and `value` to make `TOTL` and `GROW` columns:

```
pop3 %>%
  pivot_wider(
    names_from = variable,
    values_from = value
  )
#> # A tibble: 9,576 x 5
#>    country area  year   TOTL   GROW
#>    <chr>   <chr> <chr> <dbl>  <dbl>
#>  1 ABW     URB   2000  41625 1.66
#>  2 ABW     URB   2001  42025 0.956
#>  3 ABW     URB   2002  42194 0.401
#>  4 ABW     URB   2003  42277 0.197
#>  5 ABW     URB   2004  42317 0.0946
```

```
#>  6 ABW     URB   2005  42399 0.194
#>  7 ABW     URB   2006  42555 0.367
#>  8 ABW     URB   2007  42729 0.408
#>  9 ABW     URB   2008  42906 0.413
#> 10 ABW     URB   2009  43079 0.402
#> # i 9,566 more rows
```

0.4.2 Multi-choice

Based on a suggestion by Maxime Wack, https://github.com/tidyverse/tidyr/issues/384), the final example shows how to deal with a common way of recording multiple choice data. Often you will get such data as follows:

```
multi <- tribble(
  ~id, ~choice1, ~choice2, ~choice3,
  1, "A", "B", "C",
  2, "C", "B",  NA,
  3, "D",  NA,  NA,
  4, "B", "D",  NA
)
```

But the actual order isn't important, and you'd prefer to have the individual questions in the columns. You can achieve the desired transformation in two steps. First, you make the data longer, eliminating the explicit `NA`s, and adding a column to indicate that this choice was chosen:

```
multi2 <- multi %>%
  pivot_longer(
    cols = !id,
    values_drop_na = TRUE
  ) %>%
  mutate(checked = TRUE)
multi2
#> # A tibble: 8 x 4
#>      id name    value checked
#>   <dbl> <chr>   <chr> <lgl>
#> 1     1 choice1 A     TRUE
#> 2     1 choice2 B     TRUE
#> 3     1 choice3 C     TRUE
#> 4     2 choice1 C     TRUE
#> 5     2 choice2 B     TRUE
```

```
#> 6     3 choice1 D     TRUE
#> 7     4 choice1 B     TRUE
#> 8     4 choice2 D     TRUE
```

Then you make the data wider, filling in the missing observations with `FALSE`:

```
multi2 %>%
  pivot_wider(
    id_cols = id,
    names_from = value,
    values_from = checked,
    values_fill = FALSE
  )
#> # A tibble: 4 x 5
#>      id A     B     C     D
#>   <dbl> <lgl> <lgl> <lgl> <lgl>
#> 1     1 TRUE  TRUE  TRUE  FALSE
#> 2     2 FALSE TRUE  TRUE  FALSE
#> 3     3 FALSE FALSE FALSE TRUE
#> 4     4 FALSE TRUE  FALSE TRUE
```

0.5 Manual specs

The arguments to `pivot_longer()` and `pivot_wider()` allow you to pivot a wide range of datasets. But the creativity that people apply to their data structures is seemingly endless, so it's quite possible that you will encounter a dataset that you can't immediately see how to reshape with `pivot_longer()` and `pivot_wider()`. To gain more control over pivoting, you can instead create a "spec" data frame that describes exactly how data stored in the column names becomes variables (and vice versa). This section introduces you to the spec data structure, and show you how to use it when `pivot_longer()` and `pivot_wider()` are insufficient.

0.5.1 Longer

To see how this works, lets return to the simplest case of pivoting applied to the `relig_income` dataset. Now pivoting happens in two steps: we first create a spec object (using `build_longer_spec()`) then use that to describe the pivoting operation:

```
spec <- relig_income %>%
  build_longer_spec(
    cols = !religion,
```

```
    names_to = "income",
    values_to = "count"
  )
pivot_longer_spec(relig_income, spec)
#> # A tibble: 180 x 3
#>    religion income             count
#>    <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
#>  7 Agnostic $75-100k            122
#>  8 Agnostic $100-150k           109
#>  9 Agnostic >150k                84
#> 10 Agnostic Don't know/refused   96
#> # i 170 more rows
```

(This gives the same result as before, just with more code. There's no need to use it here, it is presented as a simple example for using `spec`.)

What does `spec` look like? It's a data frame with one row for each column in the wide format version of the data that is not present in the long format, and two special columns that start with `.`:

- `.name` gives the name of the column.
- `.value` gives the name of the column that the values in the cells will go into.

There is also one column in `spec` for each column present in the long format of the data that is not present in the wide format of the data. This corresponds to the `names_to` argument in `pivot_longer()` and `build_longer_spec()` and the `names_from` argument in `pivot_wider()` and `build_wider_spec()`. In this example, the income column is a character vector of the names of columns being pivoted.

```
spec
#> # A tibble: 10 x 3
#>    .name              .value income
#>    <chr>              <chr>  <chr>
#>  1 <$10k              count  <$10k
#>  2 $10-20k            count  $10-20k
#>  3 $20-30k            count  $20-30k
#>  4 $30-40k            count  $30-40k
```

```
#>  5 $40-50k            count  $40-50k
#>  6 $50-75k            count  $50-75k
#>  7 $75-100k           count  $75-100k
#>  8 $100-150k          count  $100-150k
#>  9 >150k              count  >150k
#> 10 Don't know/refused count  Don't know/refused
```

0.5.2 Wider

Below we widen `us_rent_income` with `pivot_wider()`. The result is ok, but I think it could be improved:

```
us_rent_income %>%
  pivot_wider(
    names_from = variable,
    values_from = c(estimate, moe)
  )
#> # A tibble: 52 x 6
#>    GEOID NAME                 estimate_income estimate_rent moe_income moe_rent
#>    <chr> <chr>                          <dbl>         <dbl>      <dbl>    <dbl>
#>  1 01    Alabama                        24476           747        136        3
#>  2 02    Alaska                         32940          1200        508       13
#>  3 04    Arizona                        27517           972        148        4
#>  4 05    Arkansas                       23789           709        165        5
#>  5 06    California                     29454          1358        109        3
#>  6 08    Colorado                       32401          1125        109        5
#>  7 09    Connecticut                    35326          1123        195        5
#>  8 10    Delaware                       31560          1076        247       10
#>  9 11    District of Columbia           43198          1424        681       17
#> 10 12    Florida                        25952          1077         70        3
#> # i 42 more rows
```

I think it would be better to have columns `income`, `rent`, `income_moe`, and `rent_moe`, which we can achieve with a manual spec. The current spec looks like this:

```
spec1 <- us_rent_income %>%
  build_wider_spec(
    names_from = variable,
    values_from = c(estimate, moe)
  )
spec1
```

```
#> # A tibble: 4 x 3
#>   .name           .value   variable
#>   <chr>           <chr>    <chr>
#> 1 estimate_income estimate income
#> 2 estimate_rent   estimate rent
#> 3 moe_income      moe      income
#> 4 moe_rent        moe      rent
```

For this case, we mutate `spec` to carefully construct the column names:

```
spec2 <- spec1 %>%
  mutate(
    .name = paste0(variable, ifelse(.value == "moe", "_moe", ""))
  )
spec2
#> # A tibble: 4 x 3
#>   .name      .value   variable
#>   <chr>      <chr>    <chr>
#> 1 income     estimate income
#> 2 rent       estimate rent
#> 3 income_moe moe      income
#> 4 rent_moe   moe      rent
```

Supplying this spec to `pivot_wider()` gives us the result we're looking for:

```
us_rent_income %>%
  pivot_wider_spec(spec2)
#> # A tibble: 52 x 6
#>    GEOID NAME                 income  rent income_moe rent_moe
#>    <chr> <chr>                 <dbl> <dbl>      <dbl>    <dbl>
#>  1 01    Alabama               24476   747        136        3
#>  2 02    Alaska                32940  1200        508       13
#>  3 04    Arizona               27517   972        148        4
#>  4 05    Arkansas              23789   709        165        5
#>  5 06    California            29454  1358        109        3
#>  6 08    Colorado              32401  1125        109        5
#>  7 09    Connecticut           35326  1123        195        5
#>  8 10    Delaware              31560  1076        247       10
#>  9 11    District of Columbia  43198  1424        681       17
#> 10 12    Florida               25952  1077         70        3
```

```
#> # i 42 more rows
```

0.5.3 By hand

Sometimes it's not possible (or not convenient) to compute the spec, and instead it's more convenient to construct the spec "by hand". For example, take this `construction` data, which is lightly modified from Table 5 "completions" found at https://www.census.gov/construction/nrc/index.html:

```
construction
#> # A tibble: 9 x 9
#>    Year Month  `1 unit` `2 to 4 units` `5 units or more` Northeast Midwest South
#>   <dbl> <chr>     <dbl> <lgl>                     <dbl>     <dbl>   <dbl> <dbl>
#> 1  2018 Janua~      859 NA                          348       114     169   596
#> 2  2018 Febru~      882 NA                          400       138     160   655
#> 3  2018 March       862 NA                          356       150     154   595
#> 4  2018 April       797 NA                          447       144     196   613
#> 5  2018 May         875 NA                          364        90     169   673
#> 6  2018 June        867 NA                          342        76     170   610
#> 7  2018 July        829 NA                          360       108     183   594
#> 8  2018 August      939 NA                          286        90     205   649
#> 9  2018 Septe~      835 NA                          304       117     175   560
#> # i 1 more variable: West <dbl>
```

This sort of data is not uncommon from government agencies: the column names actually belong to different variables, and here we have summaries for number of units (1, 2-4, 5+) and regions of the country (NE, NW, midwest, S, W). We can most easily describe that with a tibble:

```
spec <- tribble(
  ~.name,            ~.value, ~units,  ~region,
  "1 unit",          "n",     "1",     NA,
  "2 to 4 units",    "n",     "2-4",   NA,
  "5 units or more", "n",     "5+",    NA,
  "Northeast",       "n",     NA,      "Northeast",
  "Midwest",         "n",     NA,      "Midwest",
  "South",           "n",     NA,      "South",
  "West",            "n",     NA,      "West",
)
```

Which yields the following longer form:

```
construction %>% pivot_longer_spec(spec)
#> # A tibble: 63 x 5
#>     Year Month    units region        n
#>    <dbl> <chr>    <chr> <chr>     <dbl>
#>  1  2018 January  1     <NA>        859
#>  2  2018 January  2-4   <NA>         NA
#>  3  2018 January  5+    <NA>        348
#>  4  2018 January  <NA>  Northeast   114
#>  5  2018 January  <NA>  Midwest     169
#>  6  2018 January  <NA>  South       596
#>  7  2018 January  <NA>  West        339
#>  8  2018 February 1     <NA>        882
#>  9  2018 February 2-4   <NA>         NA
#> 10  2018 February 5+    <NA>        400
#> # i 53 more rows
```

Note that there is no overlap between the `units` and `region` variables; here the data would really be most naturally described in two independent tables.

0.5.4 Theory

One neat property of the `spec` is that you need the same spec for `pivot_longer()` and `pivot_wider()`. This makes it very clear that the two operations are symmetric:

```
construction %>%
  pivot_longer_spec(spec) %>%
  pivot_wider_spec(spec)
#> # A tibble: 9 x 9
#>    Year Month  `1 unit` `2 to 4 units` `5 units or more` Northeast Midwest South
#>   <dbl> <chr>     <dbl>          <dbl>             <dbl>     <dbl>   <dbl> <dbl>
#> 1  2018 Janua~      859             NA               348       114     169   596
#> 2  2018 Febru~      882             NA               400       138     160   655
#> 3  2018 March       862             NA               356       150     154   595
#> 4  2018 April       797             NA               447       144     196   613
#> 5  2018 May         875             NA               364        90     169   673
#> 6  2018 June        867             NA               342        76     170   610
#> 7  2018 July        829             NA               360       108     183   594
#> 8  2018 August      939             NA               286        90     205   649
#> 9  2018 Septe~      835             NA               304       117     175   560
#> # i 1 more variable: West <dbl>
```

The pivoting spec allows us to be more precise about exactly how `pivot_longer(df, spec = spec)` changes the shape of `df`: it will have `nrow(df) * nrow(spec)` rows, and `ncol(df) - nrow(spec) + ncol(spec) - 2` columns.