r4ds-ch5

Ch 5 Data tidying

5.1 Introduction

Pivoting, the primary tool for tidying data, allows you to change the form of the data without changing any value

5.1.1 Prerequisites

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

5.2 Tidy data

Each of the following datasets has the same four variables but organizes values in a different way

```
table1
```

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

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```
table3
```

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

table1 will be easiest to use in tidyverse because it's tidy

Three rules make a dataset tidy: 1. Each variable is a column; each column is a variable 2. Each observation is a row; each row is an observation 3. Each value is a cell; each cell is a value

Why make dataset tidy? 1. Consistent structure makes it easy to learn tools that work with it 2. Placing variables in columns allows R's vectorized nature to shine

dplyr, ggplot2, all other tidyverse packages are designed to work with tidy data, here are some examples

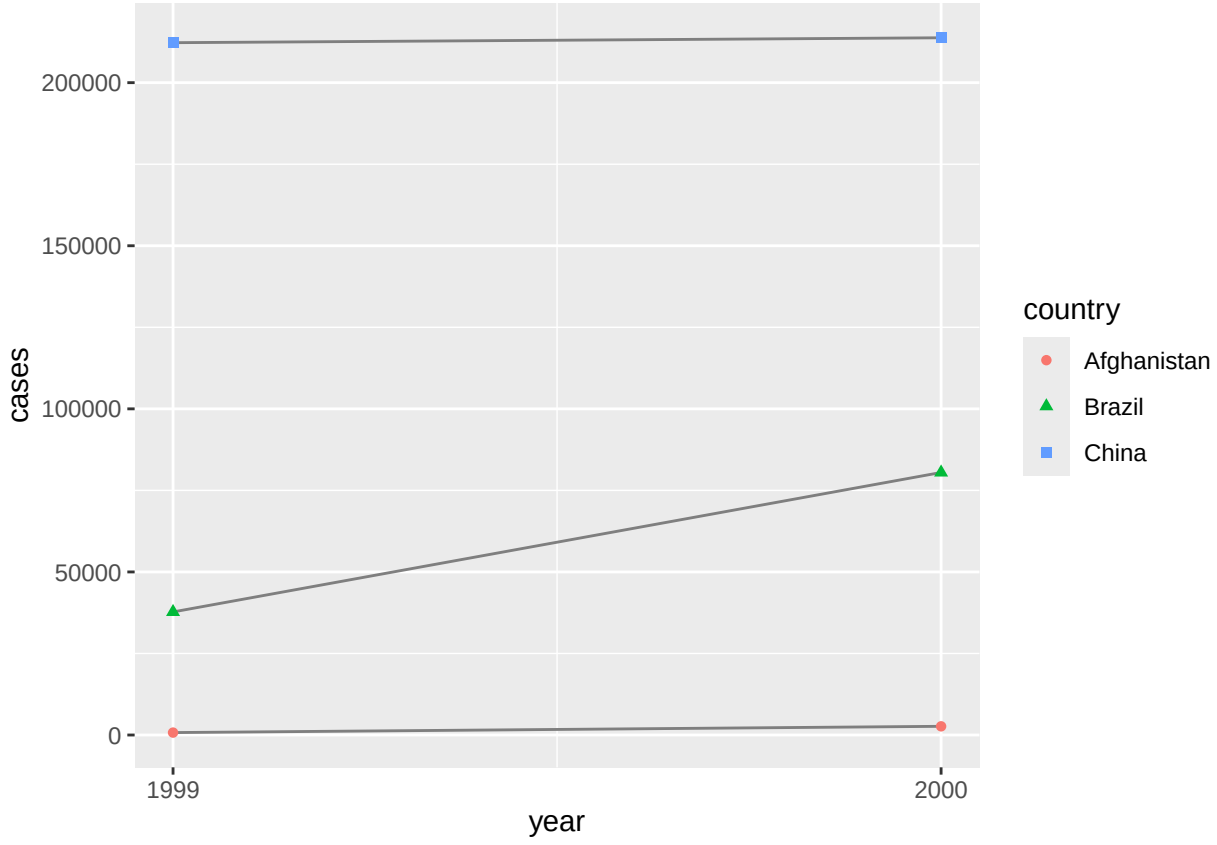
```
# Compute rate per 10,000
table1 |>
  mutate(rate = cases / population * 10000)
```

```
## # A tibble: 6 x 5
##   country      year  cases population  rate
##   <chr>       <dbl>  <dbl>      <dbl> <dbl>
## 1 Afghanistan  1999    745   19987071 0.373
## 2 Afghanistan  2000   2666   20595360 1.29
## 3 Brazil       1999  37737  172006362 2.19
## 4 Brazil       2000  80488  174504898 4.61
## 5 China        1999 212258 1272915272 1.67
## 6 China        2000 213766 1280428583 1.67
```

```r
# Compute total cases per year
table1 |>
  group_by(year) |>
  summarize(total_cases = sum(cases))
```

```
## # A tibble: 2 x 2
##    year total_cases
##   <dbl>       <dbl>
## 1  1999      250740
## 2  2000      296920
```

```r
# Visualize changes over time
ggplot(table1, aes(x = year, y = cases)) +
  geom_line(aes(group = country), color = "grey50") +
  geom_point(aes(color = country, shape = country)) +
  scale_x_continuous(breaks = c(1999, 2000))  # x-axis breaks at 1999 and 2000
```



5.2.1 Exercises

1. For each of sample tables, describe what each observation and each column represents

For table1: Each observation represents the documented number of TB cases and population for a country in a given year, and each column represents a variable of interest (country, year, cases, population)

For table2: Each observation is either the number of TB cases in a given country for a given year or the country's population in that year, and the type column represents which variable is being measured by the count column

For table3: Each observation represents the same thing as in table1, except the rate column represents two variables; each variable is not its own column

2. Sketch out the process to calculate rate (per 10k) for table2 and table3

a. Extract # of TB cases per country per year table2: Filter rows where type is cases table3: Either split the rate columns into two columns based on "/" or take whatever is to the left of "/"
b. Extract the matching population per country per year table2: Filter rows where type is population table3: Take what is to the right of "/"
c. Divide cases by population and multiply by 10000
d. Store back in appropriate place table2: Add a new row where type is called rate table3: Rate column already exists, replace it with the value computed

5.3 Lengthening data

Most real data is untidy: 1. Often structured to make entry, not analysis, easy 2. Most aren't familiar with the principles

To tidy data: 1. Figure out what underlying variables and observations are 2. Pivot data into tidy form, with variables in columns and observations in rows

tidyr provides 2 functions for pivoting: pivot_longer() and pivot_wider() pivot_longer() is most common case

5.3.1 Data in column names

billboard dataset records billboard rank of songs in year 2000

```
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>, ...
```

Each observation is a song, first 3 column are variables that describe the song Then, next 76 columns describe rank of song each week, and the column names are one variable (week) and the cells are another (rank)

To tidy, use pivot_longer()

```
billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank"
  )
```

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

After the data, there are three key arguments: 1. cols specifies which columns need to be pivoted (they aren't variables), uses same syntax as select() so could use !c(artist, track, date.entered) or starts_with("wk") 2. names_to names the variable stored in column name, which we named week 3. values_to names the variable stored in cell values, which we named rank

week and rank are in quotes because these variables don't exist yet in the data

If a song is on top 100 for less than 76 weeks, remaining weeks are filled with NAs, which don't represent unknown observations but exist because of structure of original dataset, so can ask pivot_longer() to get rid of them with values_drop_na = TRUE

```
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
```

Number of rows now much lower

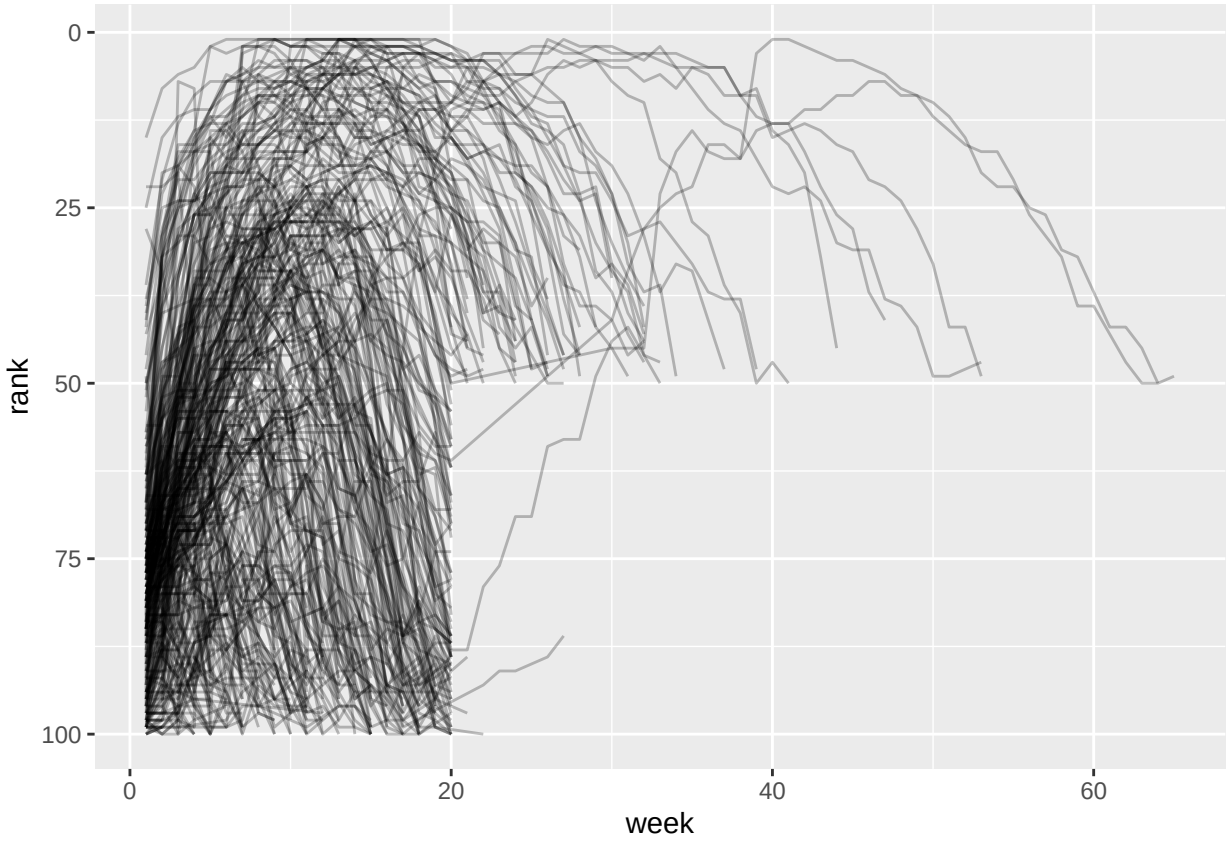
Data is now tidy but can make future computation easier by converting values of week from character strings to numbers using mutate() and readr::parse_number() parse_number() will extract the first number from the string, ignoring all other text

```
billboard_longer <- billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  ) |>
  mutate(
    week = parse_number(week)
  )
billboard_longer
```

```
## # A tibble: 5,307 x 5
##    artist  track                   date.entered  week  rank
##    <chr>   <chr>                   <date>       <dbl> <dbl>
##  1 2 Pac   Baby Don't Cry (Keep... 2000-02-26       1    87
##  2 2 Pac   Baby Don't Cry (Keep... 2000-02-26       2    82
##  3 2 Pac   Baby Don't Cry (Keep... 2000-02-26       3    72
##  4 2 Pac   Baby Don't Cry (Keep... 2000-02-26       4    77
##  5 2 Pac   Baby Don't Cry (Keep... 2000-02-26       5    87
##  6 2 Pac   Baby Don't Cry (Keep... 2000-02-26       6    94
##  7 2 Pac   Baby Don't Cry (Keep... 2000-02-26       7    99
##  8 2Ge+her The Hardest Part Of ... 2000-09-02       1    91
##  9 2Ge+her The Hardest Part Of ... 2000-09-02       2    87
## 10 2Ge+her The Hardest Part Of ... 2000-09-02       3    92
## # i 5,297 more rows
```

Now we can visualize how song ranks change over time

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



Can see that very few songs stay in top 100 for more than 20 weeks

5.3.2 How does pivoting work?

Start with simple dataset, three patient ids A, B, and C, and take two blood pressure measurements on each patient tribble() is handy way of constructing small tibbles by hand

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

Want new dataset to have three variables, id (already exists), measurement (the column names), and value (the cell values) To achieve this, need to pivot longer

```
df |>
  pivot_longer(
    cols = bp1:bp2,
    names_to = "measurement",
    values_to = "value"
  )
```

```
## # A tibble: 6 x 3
```

```
##   id    measurement value
##   <chr> <chr>       <dbl>
## 1 A     bp1           100
## 2 A     bp2           120
## 3 B     bp1           140
## 4 B     bp2           115
## 5 C     bp1           120
## 6 C     bp2           125
```

How does it work? Values in a column that are already a variable are repeated (id), once for each column that is pivoted (so twice) Column names become values (bp1, bp2) in a new variable (measurement, as defined in names_to), repeated once for each row in the dataset Cell values become new values (100, 120, etc.) in a new variable (value, as defined in values_to), unwound (not repeated) row by row

5.3.3 Many variables in column names

More challenging is when multiple pieces of info are crammed into column names, and you want to store them in separate new variables For example, take a look at who2 dataset, source of table1

```
who2
```

```
## # A tibble: 7,240 x 58
##    country      year sp_m_014 sp_m_1524 sp_m_2534 sp_m_3544 sp_m_4554 sp_m_5564
##    <chr>       <dbl>    <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
##  1 Afghanistan  1980       NA        NA        NA        NA        NA        NA
##  2 Afghanistan  1981       NA        NA        NA        NA        NA        NA
##  3 Afghanistan  1982       NA        NA        NA        NA        NA        NA
##  4 Afghanistan  1983       NA        NA        NA        NA        NA        NA
##  5 Afghanistan  1984       NA        NA        NA        NA        NA        NA
##  6 Afghanistan  1985       NA        NA        NA        NA        NA        NA
##  7 Afghanistan  1986       NA        NA        NA        NA        NA        NA
##  8 Afghanistan  1987       NA        NA        NA        NA        NA        NA
##  9 Afghanistan  1988       NA        NA        NA        NA        NA        NA
## 10 Afghanistan  1989       NA        NA        NA        NA        NA        NA
## # i 7,230 more rows
## # i 50 more variables: sp_m_65 <dbl>, sp_f_014 <dbl>, sp_f_1524 <dbl>,
## #   sp_f_2534 <dbl>, sp_f_3544 <dbl>, sp_f_4554 <dbl>, sp_f_5564 <dbl>,
## #   sp_f_65 <dbl>, sn_m_014 <dbl>, sn_m_1524 <dbl>, sn_m_2534 <dbl>,
## #   sn_m_3544 <dbl>, sn_m_4554 <dbl>, sn_m_5564 <dbl>, sn_m_65 <dbl>,
## #   sn_f_014 <dbl>, sn_f_1524 <dbl>, sn_f_2534 <dbl>, sn_f_3544 <dbl>,
## #   sn_f_4554 <dbl>, sn_f_5564 <dbl>, sn_f_65 <dbl>, ep_m_014 <dbl>, ...
```

This dataset recorded by WHO records info about TB diagnoses, two columns are variables easy to interpret (country, year), followed by 56 columns with a pattern Each is made up of three pieces separated by _, first (sp/rel/ep) describes method used for diagnosis, second (m/f) is gender (coded as binary), and third is the age range(014 being 0-14 for eaxmple) So really 6 pieces of information recorded: - country, year (already columns) - method, gender, age range (combined in other column names) - patient count (in cells)

To organize this info into 6 columns, use pivot_longer() with vector of column names for names_to and instructors for splitting original variable names into pieces for names_sep as well as a column name for values_to

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

```
## # A tibble: 405,440 x 6
##    country      year diagnosis gender age   count
##    <chr>       <dbl> <chr>     <chr>  <chr> <dbl>
##  1 Afghanistan  1980 sp        m      014      NA
##  2 Afghanistan  1980 sp        m      1524     NA
##  3 Afghanistan  1980 sp        m      2534     NA
##  4 Afghanistan  1980 sp        m      3544     NA
##  5 Afghanistan  1980 sp        m      4554     NA
##  6 Afghanistan  1980 sp        m      5564     NA
##  7 Afghanistan  1980 sp        m      65       NA
##  8 Afghanistan  1980 sp        f      014      NA
##  9 Afghanistan  1980 sp        f      1524     NA
## 10 Afghanistan  1980 sp        f      2534     NA
## # i 405,430 more rows
```

An alternative to names_sep is names_pattern, can use with regex

In the above example, instead of column names pivoting into a single column, they pivot into multiple columns Can imagine as two steps (first pivot then separate) but under hood happens in a single step

5.3.4 Data and variables names in the column headers

Next step in complexity is when column names include mix of variable values and names, for example in household dataset

```
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
```

This dataset contains data about five families, with names and date of birth of up to two children The new challenge is that the column names contain the names of two variables (dob, name), and the values of another (child, with values 1 or 2)

To solve this problem, again need to supply vector to names_to but this time use special ".value" sentinel, which isn't the name of a variable but unique value that tells pivot_longer() to do something different This overrides usual values_to argument, instead uses first component of pivoted column name as variable name in the output

```
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
```

Use values_drop_na since shape of input forces creation of explicit missing variables (such as for families with only one child)

When you use ".value" in names_to, column names in input contribute to both values and variable names in output Above, .value is used to say that whatever is before _ is the output column name and whatever is after _ is the value of the child column

5.4 Widening data

pivot_longer() tackled case where values ended up in column names; makes dataset longer by adding rows pivot_wider() makes datasets wider by adding columns and reducing rows, helps when one observation is spread across multiple rows

Start by looking at Centers of Medicare and Medicaid data about patient experiences

```
cms_patient_experience
```

```
## # A tibble: 500 x 5
##    org_pac_id org_nm                              measure_cd measure_title prf_rate
##    <chr>      <chr>                               <chr>      <chr>            <dbl>
##  1 0446157747 USC CARE MEDICAL GROUP INC          CAHPS_GRP~ CAHPS for MI~       63
##  2 0446157747 USC CARE MEDICAL GROUP INC          CAHPS_GRP~ CAHPS for MI~       87
##  3 0446157747 USC CARE MEDICAL GROUP INC          CAHPS_GRP~ CAHPS for MI~       86
##  4 0446157747 USC CARE MEDICAL GROUP INC          CAHPS_GRP~ CAHPS for MI~       57
##  5 0446157747 USC CARE MEDICAL GROUP INC          CAHPS_GRP~ CAHPS for MI~       85
##  6 0446157747 USC CARE MEDICAL GROUP INC          CAHPS_GRP~ CAHPS for MI~       24
##  7 0446162697 ASSOCIATION OF UNIVERSITY PHYSI~ CAHPS_GRP~ CAHPS for MI~       59
##  8 0446162697 ASSOCIATION OF UNIVERSITY PHYSI~ CAHPS_GRP~ CAHPS for MI~       85
##  9 0446162697 ASSOCIATION OF UNIVERSITY PHYSI~ CAHPS_GRP~ CAHPS for MI~       83
## 10 0446162697 ASSOCIATION OF UNIVERSITY PHYSI~ CAHPS_GRP~ CAHPS for MI~       63
## # i 490 more rows
```

Core unit being studied is an organization, but each is spread across 6 rows with one row for each measurement taken in survey organization

Can see complete set of values for measure_cd and measure_title by using distinct()

```
cms_patient_experience |>
  distinct(measure_cd, measure_title)
```

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

Neither columns would make good variable names, measure_cd doesn't hint at meaning of variable and measure_title includes long sentence containing spaces Use measure_cd as source for new column names now, but in real analysis might want to create own variable names that are short and meaningful

pivot_wider() has opposite interface to pivot_longer(), instead of choosing new column names need to provide existing columns that define the values (values_from) and the column name (names_from)

```
cms_patient_experience |>
  pivot_wider(
    names_from = measure_cd,
    values_from = prf_rate
  )
```

```
## # A tibble: 500 x 9
##    org_pac_id org_nm           measure_title CAHPS_GRP_1 CAHPS_GRP_2 CAHPS_GRP_3
##    <chr>      <chr>            <chr>               <dbl>       <dbl>       <dbl>
##  1 0446157747 USC CARE MEDICA~ CAHPS for MI~          63          NA          NA
##  2 0446157747 USC CARE MEDICA~ CAHPS for MI~          NA          87          NA
##  3 0446157747 USC CARE MEDICA~ CAHPS for MI~          NA          NA          86
##  4 0446157747 USC CARE MEDICA~ CAHPS for MI~          NA          NA          NA
##  5 0446157747 USC CARE MEDICA~ CAHPS for MI~          NA          NA          NA
##  6 0446157747 USC CARE MEDICA~ CAHPS for MI~          NA          NA          NA
##  7 0446162697 ASSOCIATION OF ~ CAHPS for MI~          59          NA          NA
##  8 0446162697 ASSOCIATION OF ~ CAHPS for MI~          NA          85          NA
##  9 0446162697 ASSOCIATION OF ~ CAHPS for MI~          NA          NA          83
## 10 0446162697 ASSOCIATION OF ~ CAHPS for MI~          NA          NA          NA
## # i 490 more rows
## # i 3 more variables: CAHPS_GRP_5 <dbl>, CAHPS_GRP_8 <dbl>, CAHPS_GRP_12 <dbl>
```

Output doesn't look right, still have multiple rows for each organization Need to tell pivot_wider() which column(s) have values that uniquely identify each row, in this case those are variables starting with "org"

```
cms_patient_experience |>
  pivot_wider(
    id_cols = starts_with("org"),
```

```
    names_from = measure_cd,
    values_from = prf_rate
  )
```

```
## # A tibble: 95 x 8
##    org_pac_id org_nm CAHPS_GRP_1 CAHPS_GRP_2 CAHPS_GRP_3 CAHPS_GRP_5 CAHPS_GRP_8
##    <chr>      <chr>        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
##  1 0446157747 USC C~          63          87          86          57          85
##  2 0446162697 ASSOC~          59          85          83          63          88
##  3 0547164295 BEAVE~          49          NA          75          44          73
##  4 0749333730 CAPE ~          67          84          85          65          82
##  5 0840104360 ALLIA~          66          87          87          64          87
##  6 0840109864 REX H~          73          87          84          67          91
##  7 0840513552 SCL H~          58          83          76          58          78
##  8 0941545784 GRITM~          46          86          81          54          NA
##  9 1052612785 COMMU~          65          84          80          58          87
## 10 1254237779 OUR L~          61          NA          NA          65          NA
## # i 85 more rows
## # i 1 more variable: CAHPS_GRP_12 <dbl>
```

5.4.1 How does pivot_wider() work?

Once again begin with simple dataset, this time two patients with ids A and B, three blood pressure measurements on patient A and two on patient B

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

Take the values from the value column and names from measurement column

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

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

To begin the process pivot_wider() needs to first figure out what will go in rows and columns, new column names will be unique values of measurement

```
df |>
  distinct(measurement) |>
  pull()
```

```
## [1] "bp1" "bp2" "bp3"
```

By default, rows in output are determined by all variables that aren't going into new names or values, these are called id_cols Here, there is only one column, but in general can be any number

```
df |>
  select(-measurement, -value) |>
  distinct()
```

```
## # A tibble: 2 x 1
##   id
##   <chr>
## 1 A
## 2 B
```

pivot_wider() then combines these results to generate an empty dataframe

```
df |>
  select(-measurement, -value) |>
  distinct() |>
  mutate(x = NA, y = NA, z = NA)
```

```
## # A tibble: 2 x 4
##   id    x     y     z
##   <chr> <lgl> <lgl> <lgl>
## 1 A     NA    NA    NA
## 2 B     NA    NA    NA
```

It then fills in all the missing values using data in the input, in this case not every cell has corresponding value in input as there's no third bp measurement for patient B, so that cell remains missing

What happens if there are multiple rows in the input that correspond to one cell in the output? The example below has two rows that correspond to id "A" and measurement "bp1"

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

If attempt to pivot this, get an output that contains list-columns

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

```
## Warning: Values from `value` are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = {summary_fun}` to summarise duplicates.
## * Use the following dplyr code to identify duplicates.
##   {data} |>
##   dplyr::summarise(n = dplyr::n(), .by = c(id, measurement)) |>
##   dplyr::filter(n > 1L)
```

```
## # A tibble: 2 x 3
##   id    bp1       bp2
##   <chr> <list>    <list>
## 1 A     <dbl [2]> <dbl [1]>
## 2 B     <dbl [1]> <dbl [1]>
```

Follow the hint to figure out where the problem is

```
df |>
  group_by(id, measurement) |>
  summarize(n = n(), .groups = "drop") |>
  filter(n > 1)
```

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

Then up to you to figure out what's gone wrong and repair underlying damage or use grouping/summarizing skills to ensure each combo of row and column values only has a single row

5.5 Summary

Examples here are selection of those from vignette("pivot", package = "tidyr") so go there if encounter a problem this chapter doesn't cover

Another challenge is it can be hard to define what a variable is, fine to be pragmatic and use whatever makes analysis easiest