Skills Problem Set IV

Earnest Salgado

05/04/2021

```
rm(list = ls())
library(tidyverse)
library(tidyr)
```

This submission is my work alone and complies with the 30535 integrity policy.

Add your initials to indicate your agreement: **ES**

Add your collaborators: **GATW**

Late coins used this pset: 0. Late coins left: 9.

1 Tidy

1.1 Tidy data with pivot_wider() and pivot_longer() (25 points)

1. The data set billboard inside tidyr. Has the song rankings for Billboard top 100 in the year 2000. Is this data tidy? If not, identify the problem and solve it. Be careful with missing values, do we need them in the final data set or not?

There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

As we observe billboard, it does not meet these three criteria so we cannot consider it tidy. In this dataset, the column header are the values: the week numbers. This violates 1 and 2 above. We need to assign the week numbers a column of its own and the values, the ranks, a column of it own. To tidy this dataset, we first gather together all the week columns and the column names give the week and the values become the ranks.

In regards to missing values, we use na.rm to drop the missing values from the gather columns. The missing values represent weeks that the song wasn't a part of the charts on Billboard 100, so they can be safely dropped.

```
billboard %>% as_tibble()
```

```
## # 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 D~ 2000-02-26      87    82    72    77    87    94    99    NA
##  2 2Ge+her  The Ha~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
##  3 3 Doors~ Krypto~ 2000-04-08      81    70    68    67    66    57    54    53
##  4 3 Doors~ Loser   2000-10-21      76    76    72    69    67    65    55    59
##  5 504 Boyz Wobble~ 2000-04-15      57    34    25    17    17    31    36    49
##  6 98^0     Give M~ 2000-08-19      51    39    34    26    26    19     2     2
##  7 A*Teens  Dancin~ 2000-07-08      97    97    96    95   100    NA    NA    NA
##  8 Aaliyah  I Don't~ 2000-01-29      84    62    51    41    38    35    35    38
##  9 Aaliyah  Try Ag~ 2000-03-18      59    53    38    28    21    18    16    14
## 10 Adams, ~ Open M~ 2000-08-26      76    76    74    69    68    67    61    58
## # ... with 307 more rows, and 68 more variables: wk9 <dbl>, wk10 <dbl>,
## #   wk11 <dbl>, wk12 <dbl>, wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>,
## #   wk17 <dbl>, wk18 <dbl>, wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>,
## #   wk23 <dbl>, wk24 <dbl>, wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>,
## #   wk29 <dbl>, wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>,
## #   wk35 <dbl>, wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>,
## #   wk41 <dbl>, wk42 <dbl>, wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>,
## #   wk47 <dbl>, wk48 <dbl>, wk49 <dbl>, wk50 <dbl>, wk51 <dbl>, wk52 <dbl>,
## #   wk53 <dbl>, wk54 <dbl>, wk55 <dbl>, wk56 <dbl>, wk57 <dbl>, wk58 <dbl>,
## #   wk59 <dbl>, wk60 <dbl>, wk61 <dbl>, wk62 <dbl>, wk63 <dbl>, wk64 <dbl>,
## #   wk65 <dbl>, wk66 <lgl>, wk67 <lgl>, wk68 <lgl>, wk69 <lgl>, wk70 <lgl>,
## #   wk71 <lgl>, wk72 <lgl>, wk73 <lgl>, wk74 <lgl>, wk75 <lgl>, wk76 <lgl>
```

```
glimpse(billboard)
```

```
## Rows: 317
## Columns: 79
## $ artist       <chr> "2 Pac", "2Ge+her", "3 Doors Down", "3 Doors Down", "504 ~
## $ track        <chr> "Baby Don't Cry (Keep...", "The Hardest Part Of ...", "Kr~
## $ date.entered <date> 2000-02-26, 2000-09-02, 2000-04-08, 2000-10-21, 2000-04-~
## $ wk1          <dbl> 87, 91, 81, 76, 57, 51, 97, 84, 59, 76, 84, 57, 50, 71, 7~
## $ wk2          <dbl> 82, 87, 70, 76, 34, 39, 97, 62, 53, 76, 84, 47, 39, 51, 6~
## $ wk3          <dbl> 72, 92, 68, 72, 25, 34, 96, 51, 38, 74, 75, 45, 30, 28, 5~
## $ wk4          <dbl> 77, NA, 67, 69, 17, 26, 95, 41, 28, 69, 73, 29, 28, 18, 4~
## $ wk5          <dbl> 87, NA, 66, 67, 17, 26, 100, 38, 21, 68, 73, 23, 21, 13, ~
## $ wk6          <dbl> 94, NA, 57, 65, 31, 19, NA, 35, 18, 67, 69, 18, 19, 13, 3~
## $ wk7          <dbl> 99, NA, 54, 55, 36, 2, NA, 35, 16, 61, 68, 11, 20, 11, 34~
## $ wk8          <dbl> NA, NA, 53, 59, 49, 2, NA, 38, 14, 58, 65, 9, 17, 1, 29, ~
## $ wk9          <dbl> NA, NA, 51, 62, 53, 3, NA, 38, 12, 57, 73, 9, 17, 1, 27, ~
## $ wk10         <dbl> NA, NA, 51, 61, 57, 6, NA, 36, 10, 59, 83, 11, 17, 2, 30,~
## $ wk11         <dbl> NA, NA, 51, 61, 64, 7, NA, 37, 9, 66, 92, 1, 17, 2, 36, N~
## $ wk12         <dbl> NA, NA, 51, 59, 70, 22, NA, 37, 8, 68, NA, 1, 3, 3, 37, N~
## $ wk13         <dbl> NA, NA, 47, 61, 75, 29, NA, 38, 6, 61, NA, 1, 3, 3, 39, N~
## $ wk14         <dbl> NA, NA, 44, 66, 76, 36, NA, 49, 1, 67, NA, 1, 7, 4, 49, N~
## $ wk15         <dbl> NA, NA, 38, 72, 78, 47, NA, 61, 2, 59, NA, 4, 10, 12, 57,~
## $ wk16         <dbl> NA, NA, 28, 76, 85, 67, NA, 63, 2, 63, NA, 8, 17, 11, 63,~
## $ wk17         <dbl> NA, NA, 22, 75, 92, 66, NA, 62, 2, 67, NA, 12, 25, 13, 65~
## $ wk18         <dbl> NA, NA, 18, 67, 96, 84, NA, 67, 2, 71, NA, 22, 29, 15, 68~
## $ wk19         <dbl> NA, NA, 18, 73, NA, 93, NA, 83, 3, 79, NA, 23, 29, 18, 79~
## $ wk20         <dbl> NA, NA, 14, 70, NA, 94, NA, 86, 4, 89, NA, 43, 40, 20, 86~
## $ wk21         <dbl> NA, NA, 12, NA, NA, NA, NA, NA, 5, NA, NA, 44, 43, 30, NA~
## $ wk22         <dbl> NA, NA, 7, NA, NA, NA, NA, NA, 5, NA, NA, NA, 50, 40, NA,~
## $ wk23         <dbl> NA, NA, 6, NA, NA, NA, NA, NA, 6, NA, NA, NA, NA, 39, NA,~
```

```
## $ wk24         <dbl> NA, NA, 6, NA, NA, NA, NA, NA, 9, NA, NA, NA, NA, 44, NA,~
## $ wk25         <dbl> NA, NA, 6, NA, NA, NA, NA, NA, 13, NA, NA, NA, NA, NA, NA~
## $ wk26         <dbl> NA, NA, 5, NA, NA, NA, NA, NA, 14, NA, NA, NA, NA, NA, NA~
## $ wk27         <dbl> NA, NA, 5, NA, NA, NA, NA, NA, 16, NA, NA, NA, NA, NA, NA~
## $ wk28         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 23, NA, NA, NA, NA, NA, NA~
## $ wk29         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 22, NA, NA, NA, NA, NA, NA~
## $ wk30         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 33, NA, NA, NA, NA, NA, NA~
## $ wk31         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 36, NA, NA, NA, NA, NA, NA~
## $ wk32         <dbl> NA, NA, 3, NA, NA, NA, NA, NA, 43, NA, NA, NA, NA, NA, NA~
## $ wk33         <dbl> NA, NA, 3, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk34         <dbl> NA, NA, 3, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk35         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk36         <dbl> NA, NA, 5, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk37         <dbl> NA, NA, 5, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk38         <dbl> NA, NA, 9, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk39         <dbl> NA, NA, 9, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ wk40         <dbl> NA, NA, 15, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk41         <dbl> NA, NA, 14, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk42         <dbl> NA, NA, 13, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk43         <dbl> NA, NA, 14, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk44         <dbl> NA, NA, 16, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk45         <dbl> NA, NA, 17, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk46         <dbl> NA, NA, 21, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk47         <dbl> NA, NA, 22, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk48         <dbl> NA, NA, 24, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk49         <dbl> NA, NA, 28, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk50         <dbl> NA, NA, 33, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk51         <dbl> NA, NA, 42, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk52         <dbl> NA, NA, 42, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk53         <dbl> NA, NA, 49, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk54         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk55         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk56         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk57         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk58         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk59         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk60         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk61         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk62         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk63         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk64         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk65         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk66         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk67         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk68         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk69         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk70         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk71         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk72         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk73         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk74         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk75         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ wk76         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

```
library(dplyr)
billboard.newdf <- billboard %>% gather(week, rank, wk1:wk76, na.rm = TRUE)
billboard.newdf <- arrange(billboard.newdf, artist, track)
```

```
billboard.df <- billboard.newdf %>% mutate(
  week = parse_number(week),
  date = as.Date(date.entered) + 7*(week - 1))

billboard.df <- billboard.df[c("artist", "track", "date", "week", "rank")]
head(billboard.df)
```

```
## # A tibble: 6 x 5
##   artist track                   date        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... 2000-03-04     2    82
## 3 2 Pac  Baby Don't Cry (Keep... 2000-03-11     3    72
## 4 2 Pac  Baby Don't Cry (Keep... 2000-03-18     4    77
## 5 2 Pac  Baby Don't Cry (Keep... 2000-03-25     5    87
## 6 2 Pac  Baby Don't Cry (Keep... 2000-04-01     6    94
```

2. The data set fish_encounters inside tidyr shows information about different monitors that capture fish swimming down a river.
3. Is this data tidy? If not, identify the problem and solve it.

No, the data is not tidy. The column headers are not variables, and the variables listed under 'Station' are scattered across multiple rows. Under the column fish, it does show ID numbers for different tagged fish, but we could not see from the initial data set up how the fish is monitored by station as it swims down the river. We can use pivot_wider to fix this below:

```
fish_encounters %>% as_tibble()
```

```
## # 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
## # ... with 104 more rows
```

```
glimpse(fish_encounters)
```

```
## Rows: 114
## Columns: 3
```

```
## $ fish    <fct> 4842, 4842, 4842, 4842, 4842, 4842, 4842, 4842, 4842, 4842, 48~
## $ station <fct> Release, I80_1, Lisbon, Rstr, Base_TD, BCE, BCW, BCE2, BCW2, M~
## $ seen    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
```

```
sapply(fish_encounters, class)
```

```
##      fish   station      seen
##  "factor"  "factor" "integer"
```

```
fish_encounters$fish <- as.character(fish_encounters$fish)
encounters_fish <- fish_encounters %>%
  pivot_wider(names_from = station, values_from = seen)
encounters_fish[is.na(encounters_fish)] <- 0
```

2. Which kind of missing values does this data has? What do they mean?

In regards to missing values, we use is.na to populate the missing values with zeroes. The missing values represent when the fish was not detected at that particular station.

3. The data set us_rent_income inside tidyr shows income and rent in 2017 from the American Community Survey. Is this data tidy? If not, identify the problem and solve it. How is this case different from the ones we have seen so far?

This dataset as well is also unsurprisingly, not tidy. Essentially each observation (for example income and rent estimates for Alabama) is on multiple rows, and the variable column is not a true variable in that it serves as just a header for the true variables, income and rent. We can also use pivot_wider to fix the data here.

```
us_rent_income %>% as_tibble()
```

```
## # 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
## # ... with 94 more rows
```

```
glimpse(us_rent_income)
```

```
## Rows: 104
## Columns: 5
## $ GEOID    <chr> "01", "01", "02", "02", "04", "04", "05", "05", "06", "06", "~
## $ NAME     <chr> "Alabama", "Alabama", "Alaska", "Alaska", "Arizona", "Arizona~
## $ variable <chr> "income", "rent", "income", "rent", "income", "rent", "income~
## $ estimate <dbl> 24476, 747, 32940, 1200, 27517, 972, 23789, 709, 29454, 1358,~
## $ moe      <dbl> 136, 3, 508, 13, 148, 4, 165, 5, 109, 3, 109, 5, 195, 5, 247,~
```

```
sapply(us_rent_income, class)
```

```
##       GEOID        NAME    variable    estimate         moe
## "character" "character" "character"   "numeric"   "numeric"
```

```
new_usrentincome <- us_rent_income %>%
  select(-moe) %>%
  pivot_wider(names_from = variable, values_from = estimate) %>%
  drop_na()
```

4. pivot_longer() and pivot_wider() are not perfectly symmetrical. Carefully consider the following example. Why do we need quotes on the arguments names_to and values_to, but not in names_from and values_from?

The general rule is if you're identifying an existing column (e.g., game), do not quote. If you're talking about a column that does not currently exist (e.g., player in new tibble), quote it. Thus, We do not have to quote arguments for names_from and values_from because they already exist in our data. You have to quote any argument to values_to because its referencing a column that does not exist.

```
soccer <- tibble(
  game = c("Real Sociedad", "Real Sociedad", "Huesca", "Huesca"),
  player = c("Messi", "Griezmann", "Messi", "Griezmann"),
  goals = c(2,1,2,1)
)
soccer %>%
  pivot_wider(names_from = player, values_from = goals) %>%
  pivot_longer(Messi:Griezmann,
               names_to = "player",
               values_to = "goals")
```

```
## # A tibble: 4 x 3
##   game          player    goals
##   <chr>         <chr>     <dbl>
## 1 Real Sociedad Messi         2
## 2 Real Sociedad Griezmann     1
## 3 Huesca        Messi         2
## 4 Huesca        Griezmann     1
```

5. This code fails. Explain the error message. How could it be fixed?

The code fails because tidyverse functions will interpret numbers without quotes or backticks, like 1999 and 2000, as column numbers. In this case, pivot_longer() tries to select the 1999th and 2000th column of the data frame. To select the columns 1999 and 2000, the names must be surrounded in backticks (‘) or as strings.

```
table4a %>%
  pivot_longer(‘1999‘:‘2000‘,
               names_to = "year",
               values_to = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

6. Why does pivot_wider fails on this tibble? Add a new column to address the problem and show that pivot_wider works on your new update dataset.

Widening this data frame using pivot_wider() produces columns that are lists of numeric vectors because the name and key columns do not uniquely identify rows. In particular, there are two rows with values for the goals of "Messi".

We could solve the problem by adding a row with a distinct observation count for each combination of player and game.

```
soccer <- tribble(
  ~player, ~game, ~goals,
  "Messi", "Real Sociedad", 2,
  "Messi", "Huesca", 2,
  "Messi", "Real Sociedad", 0,
  "Messi", "Huesca", 1,
  "Griezmann", "Real Sociedad", 1,
  "Griezmann", "Huesca", 1
)
football <- soccer %>%
  group_by(player, game) %>%
  mutate(obs = row_number()) %>%
  pivot_wider(names_from = game, values_from = goals)
```

7. Tidy the pivot table below. Do you need to make it wider or longer? What are the variables?

We should use pivot_longer() to create a longer table. Specifically, we can combine "female" and "male" into one variable, "sex". Thus, we would have three variables that are unique combinations of sex and pregnancy status: pregnant, sex, and count.

Since males will never have a yes count for pregnancy, we can simply remove this observation.

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes", NA, 10,
  "no", 20, 12
)
longer_tidy_preg <- preg %>%
  pivot_longer(c(male, female), names_to = "sex", values_to = "count", values_drop_na = TRUE) %>%
  arrange(count)
longer_tidy_preg
```

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

```
##   pregnant sex    count
##   <chr>    <chr>  <dbl>
## 1 yes      female    10
## 2 no       female    12
## 3 no       male      20
```

Conversely, we can also use gather() the sex variable. In terms of style and being concise, I would prefer this code syntax.

```
preg %>%
  gather(male, female, key="gender", value="Number")
```

```
## # A tibble: 4 x 3
##   pregnant gender Number
##   <chr>    <chr>   <dbl>
## 1 yes      male       NA
## 2 no       male       20
## 3 yes      female     10
## 4 no       female     12
```

8. What do the extra and fill arguments do in separate()? Hint: experiment with the various options for the following two data sets

The extra argument tells separate() what to do if there are additional pieces, and the fill argument tells it what to do if there are less pieces than expected. By default, separate() drops extra values with a warning.

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"))
```

```
## Warning: Expected 3 pieces. Additional pieces discarded in 1 rows [2].
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f
## 3 h     i     j
```

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"))
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1 rows [2].
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     <NA>
## 3 f     g     i
```

Adding the argument, extra = "drop", gives same result as above without the warning message.

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"), extra = "drop")
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f
## 3 h     i     j
```

Further experimenting with extras, extra = "merge" gives the extra values unsplit, so "f,g" appears in column three.

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"), extra = "merge")
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f,g
## 3 h     i     j
```

The default for fill is similar to those in separate(); it fills columns with missing values but emits a warning. In this example, the 2nd row of column three is NA.

Alternative options for fill are "right", to fill with missing values from the right, but without a warning. The same goes for fill = "left", except missing values are filled from the left and the NA value shifts accordingly.

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), fill = "right")
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     <NA>
## 3 f     g     i
```

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), fill = "left")
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 <NA>  d     e
## 3 f     g     i
```

1.2 tidying case study (30 pts)

1. In this WHO case study in Ch. 12.6 Hadley set na.rm = TRUE just to make it easier to check that we had the correct values.
2. Are there implicit missing values? Use a command you learned in the tidy data slides/videos. If there are implicit missing values, how many rows? If not, show how you know that there are not.

By first looking at the who dataset, we observe there is a lot of potential variables to unpack just under the "key" variable. We can claim there is implicit missing values by saying this dataset doesn't have any reported cases for males, or by age.

First, lets tidy the data and introduce a key to capture the column names that are actually values.

```
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> <int>       <int>        <int>        <int>        <int>
##  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
## # ... with 7,230 more rows, and 52 more variables: new_sp_m4554 <int>,
## #   new_sp_m5564 <int>, new_sp_m65 <int>, new_sp_f014 <int>,
## #   new_sp_f1524 <int>, new_sp_f2534 <int>, new_sp_f3544 <int>,
## #   new_sp_f4554 <int>, new_sp_f5564 <int>, new_sp_f65 <int>,
## #   new_sn_m014 <int>, new_sn_m1524 <int>, new_sn_m2534 <int>,
## #   new_sn_m3544 <int>, new_sn_m4554 <int>, new_sn_m5564 <int>,
## #   new_sn_m65 <int>, new_sn_f014 <int>, new_sn_f1524 <int>,
## #   new_sn_f2534 <int>, new_sn_f3544 <int>, new_sn_f4554 <int>,
## #   new_sn_f5564 <int>, new_sn_f65 <int>, new_ep_m014 <int>,
## #   new_ep_m1524 <int>, new_ep_m2534 <int>, new_ep_m3544 <int>,
## #   new_ep_m4554 <int>, new_ep_m5564 <int>, new_ep_m65 <int>,
## #   new_ep_f014 <int>, new_ep_f1524 <int>, new_ep_f2534 <int>,
## #   new_ep_f3544 <int>, new_ep_f4554 <int>, new_ep_f5564 <int>,
## #   new_ep_f65 <int>, newrel_m014 <int>, newrel_m1524 <int>,
## #   newrel_m2534 <int>, newrel_m3544 <int>, newrel_m4554 <int>,
## #   newrel_m5564 <int>, newrel_m65 <int>, newrel_f014 <int>,
## #   newrel_f1524 <int>, newrel_f2534 <int>, newrel_f3544 <int>,
## #   newrel_f4554 <int>, newrel_f5564 <int>, newrel_f65 <int>
```

```
who_tidy <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
```

```
  )
who_tidy
```

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key          cases
##    <chr>       <chr> <chr> <int> <chr>        <int>
##  1 Afghanistan AF    AFG    1997 new_sp_m014      0
##  2 Afghanistan AF    AFG    1997 new_sp_m1524    10
##  3 Afghanistan AF    AFG    1997 new_sp_m2534     6
##  4 Afghanistan AF    AFG    1997 new_sp_m3544     3
##  5 Afghanistan AF    AFG    1997 new_sp_m4554     5
##  6 Afghanistan AF    AFG    1997 new_sp_m5564     2
##  7 Afghanistan AF    AFG    1997 new_sp_m65       0
##  8 Afghanistan AF    AFG    1997 new_sp_f014      5
##  9 Afghanistan AF    AFG    1997 new_sp_f1524    38
## 10 Afghanistan AF    AFG    1997 new_sp_f2534    36
## # ... with 76,036 more rows
```

Using na.rm = TRUE depends on how the missing values are represented in this dataset. We'd like to determine if a missing value represents that there were no cases of TB or whether it means that the WHO does not have any data on total number of TB cases. We know from Chapter 12.5 in the textbook that if there are no explicit 0 values in the data, then missing values may be used to indicate no cases. However, if there are both explicit and implicit missing values, then it suggests that missing values are being used to represent different things. If that were the case it is likely that explicit missing values would mean none or zero cases, and implicit missing values would mean no data on the number of cases.

First, we check for the presence of zeros in the data.

```
who_tidy %>%
  filter(cases == 0) %>%
  nrow()
```

```
## [1] 11080
```

2. How many country-year pairs are explicitly missing TB data? To answer this question, it's beneficial to check if all values for a (country, year) are missing, or simply just a portion of the values.

```
pivot_longer(who, c(new_sp_m014:newrel_f65), names_to = "key", values_to = "cases") %>%
  group_by(country, year) %>%
  mutate(prop_missing = sum(is.na(cases)) / n()) %>%
  filter(prop_missing > 0, prop_missing < 1)
```

```
## # A tibble: 195,104 x 7
## # Groups:   country, year [3,484]
##    country     iso2  iso3   year key          cases prop_missing
##    <chr>       <chr> <chr> <int> <chr>        <int>        <dbl>
##  1 Afghanistan AF    AFG    1997 new_sp_m014      0         0.75
##  2 Afghanistan AF    AFG    1997 new_sp_m1524    10         0.75
##  3 Afghanistan AF    AFG    1997 new_sp_m2534     6         0.75
##  4 Afghanistan AF    AFG    1997 new_sp_m3544     3         0.75
##  5 Afghanistan AF    AFG    1997 new_sp_m4554     5         0.75
##  6 Afghanistan AF    AFG    1997 new_sp_m5564     2         0.75
```

```
##  7 Afghanistan AF    AFG    1997 new_sp_m65       0         0.75
##  8 Afghanistan AF    AFG    1997 new_sp_f014      5         0.75
##  9 Afghanistan AF    AFG    1997 new_sp_f1524    38         0.75
## 10 Afghanistan AF    AFG    1997 new_sp_f2534    36         0.75
## # ... with 195,094 more rows
```

From the results above, it looks like it is possible for a (country, year) row to contain only some of its total missing values.

2. In this WHO case study in Ch.12.6, what's the difference between an NA and zero?

The presence of these zeros in the who dataset shows that cases of zero TB are explicitly entered in the data as "0", and missing values "NA" represents missing data about TB cases.

We also can check for implicit missing values. Implicit missing values are (year, country) combinations that do not appear in the data.

```
nrow(who)
```

```
## [1] 7240
```

```
who %>%
  complete(country, year) %>%
  nrow()
```

```
## [1] 7446
```

Since the number of complete cases of (country, year) is greater than the number of rows in who, there are some implicit values.

OK, but how can we identify these implicit values? We employ the anti_join() function:

```
anti_join(complete(who, country, year), who, by = c("country", "year")) %>%
  select(country, year) %>%
  group_by(country) %>%
  summarise(min_year = min(year), max_year = max(year))
```

```
## # A tibble: 9 x 3
##   country                           min_year max_year
##   <chr>                                <int>    <int>
## 1 Bonaire, Saint Eustatius and Saba     1980     2009
## 2 Curacao                               1980     2009
## 3 Montenegro                            1980     2004
## 4 Netherlands Antilles                  2010     2013
## 5 Serbia                                1980     2004
## 6 Serbia & Montenegro                   2005     2013
## 7 Sint Maarten (Dutch part)             1980     2009
## 8 South Sudan                           1980     2010
## 9 Timor-Leste                           1980     2001
```

All of these refer to (country, year) combinations for years prior to the existence of the actual country. For example, Timor-Leste achieved independence in 2002, so years prior to that are not included in the data.

To summarize:

0 = no cases of TB. Explicit missing values (NAs) = missing data for (country, year) combinations in which the country existed in that year. Implicit missing values represent missing data in periods when a particular country did not exist in that year.

3. What happens if you neglect the mutate() step?

This question refers to missing a step while tidying the who dataset. Technically, by neglecting the mutate step you will have one less column in the data instead of a split between "new" and "type".

```
who_tidy2 <- who_tidy %>%
  mutate(names_from = stringr::str_replace(key, "newrel", "new_rel"))
who_tidy3 <- who_tidy2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2580 rows [243,
## 244, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 903,
## 904, 905, 906, ...].
```

If we filter our new dataframe for "newrel_", we see that sexage is missing completely, and type = m014.

```
who_tidycheck <- who_tidy %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2580 rows [243,
## 244, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 903,
## 904, 905, 906, ...].
```

```
filter(who_tidycheck, new == "newrel") %>% head()
```

```
## # A tibble: 6 x 8
##   country     iso2  iso3   year new    type  sexage cases
##   <chr>       <chr> <chr> <int> <chr>  <chr> <chr>  <int>
## 1 Afghanistan AF    AFG    2013 newrel m014  <NA>    1705
## 2 Afghanistan AF    AFG    2013 newrel f014  <NA>    1749
## 3 Albania     AL    ALB    2013 newrel m014  <NA>      14
## 4 Albania     AL    ALB    2013 newrel m1524 <NA>      60
## 5 Albania     AL    ALB    2013 newrel m2534 <NA>      61
## 6 Albania     AL    ALB    2013 newrel m3544 <NA>      32
```

4. Health outcomes are often sexed. As in certain maladies are more associated with males or females. Using the tidied WHO data, you will make an informative visualization to address the question: "To what extent is Tuberculosis associated with a specific sex and has this changed from 1997 onward?"
5. For each country, year, and sex compute the total number of cases of TB.

```
who_tidy4 <- who_tidy3 %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)

who_tidy4 %>%
  group_by(country, year, sex) %>%
  filter(year > 1995) %>%
  summarise(cases = sum(cases)) %>%
  unite(country_sex, country, sex, remove = FALSE)
```

```
## # A tibble: 6,402 x 5
## # Groups:   country, year [3,320]
##    country_sex   country      year sex   cases
##    <chr>         <chr>       <int> <chr> <int>
##  1 Afghanistan_f Afghanistan  1997 f       102
##  2 Afghanistan_m Afghanistan  1997 m        26
##  3 Afghanistan_f Afghanistan  1998 f      1207
##  4 Afghanistan_m Afghanistan  1998 m       571
##  5 Afghanistan_f Afghanistan  1999 f       517
##  6 Afghanistan_m Afghanistan  1999 m       228
##  7 Afghanistan_f Afghanistan  2000 f      1751
##  8 Afghanistan_m Afghanistan  2000 m       915
##  9 Afghanistan_f Afghanistan  2001 f      3062
## 10 Afghanistan_m Afghanistan  2001 m      1577
## # ... with 6,392 more rows
```

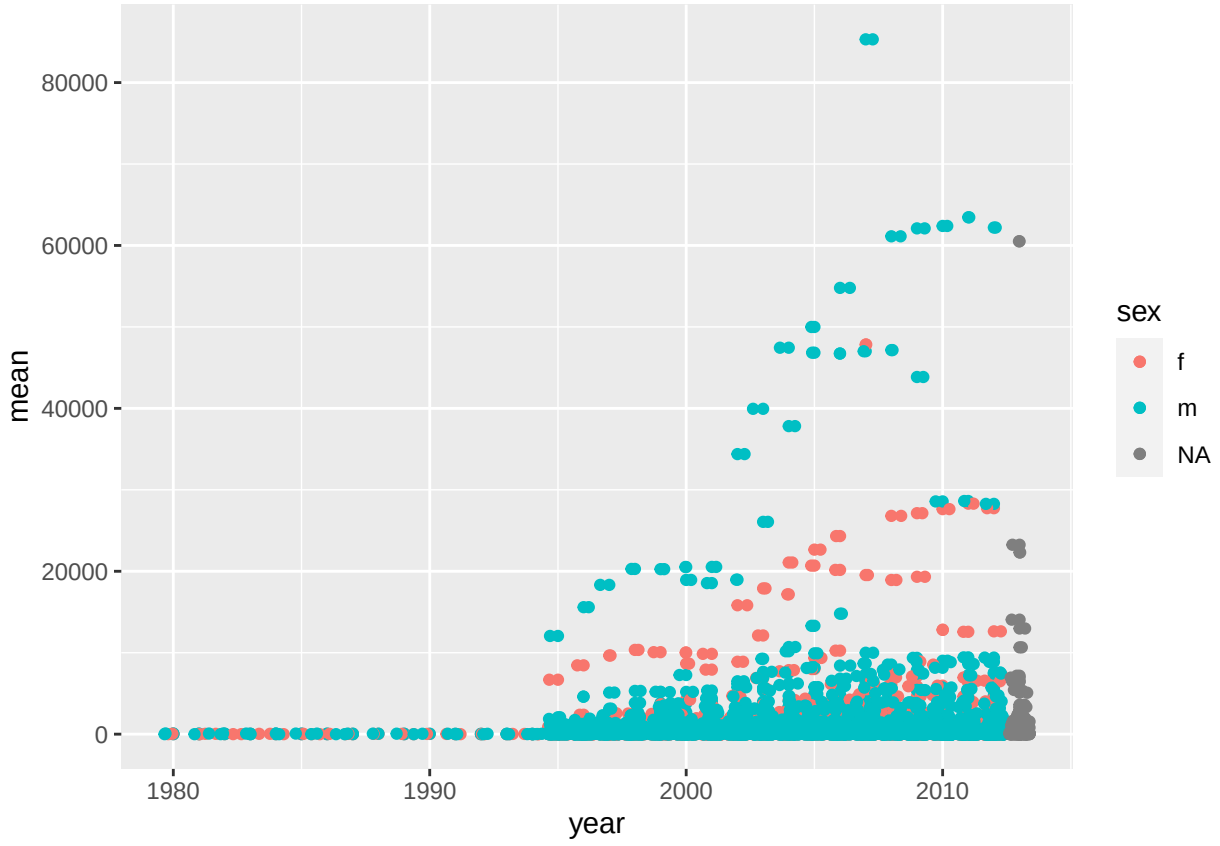
2. Using raw values is probably not going to provide clear evidence. Why not?

To adequately address the question: "To what extent is Tuberculosis associated with a specific sex and has this changed from 1997 onward?" we will need to bring in a plot to visualize the data and pull from it clear associations.

3. For each country-year, compute the ratio of male to female patients.

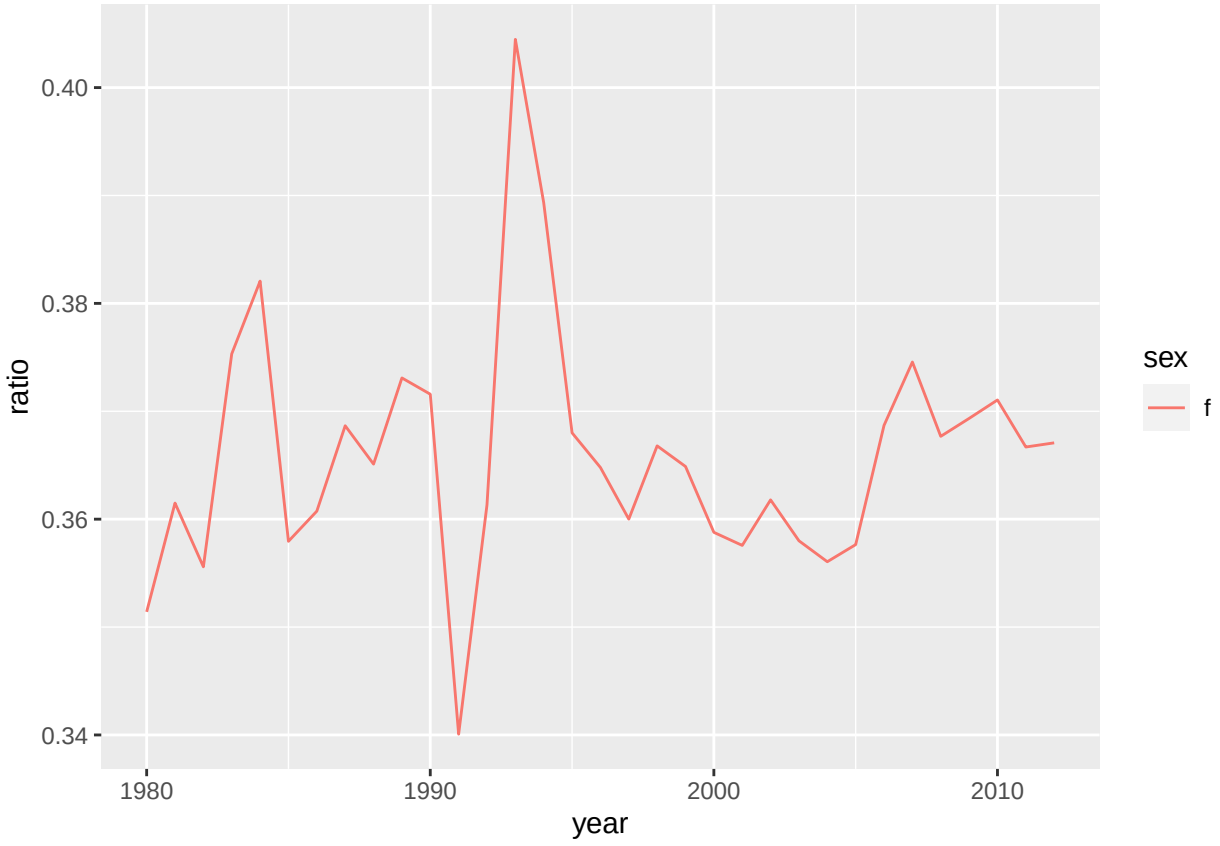
```
who_tidy4 %>%
  group_by(sex, year, country) %>%
  summarise(mean=mean(cases)) %>%
  ggplot(aes(x=year, y=mean, colour=sex))+
  geom_point()+
  geom_jitter()
```

```
## `summarise()` has grouped output by 'sex', 'year'. You can override using the `.groups` argument.
```




```
who_tidy4 %>%
  group_by(sex, year) %>%
  summarise(meansex=sum(cases)) %>%
  ungroup() %>%
  group_by(year) %>%
  mutate(tot=sum(meansex)) %>%
  ungroup() %>%
  mutate(ratio=meansex/tot) %>%
  filter(sex=="f") %>%
  ggplot(aes(x=year, y=ratio, colour=sex))+
  geom_line()
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups` argument.
```



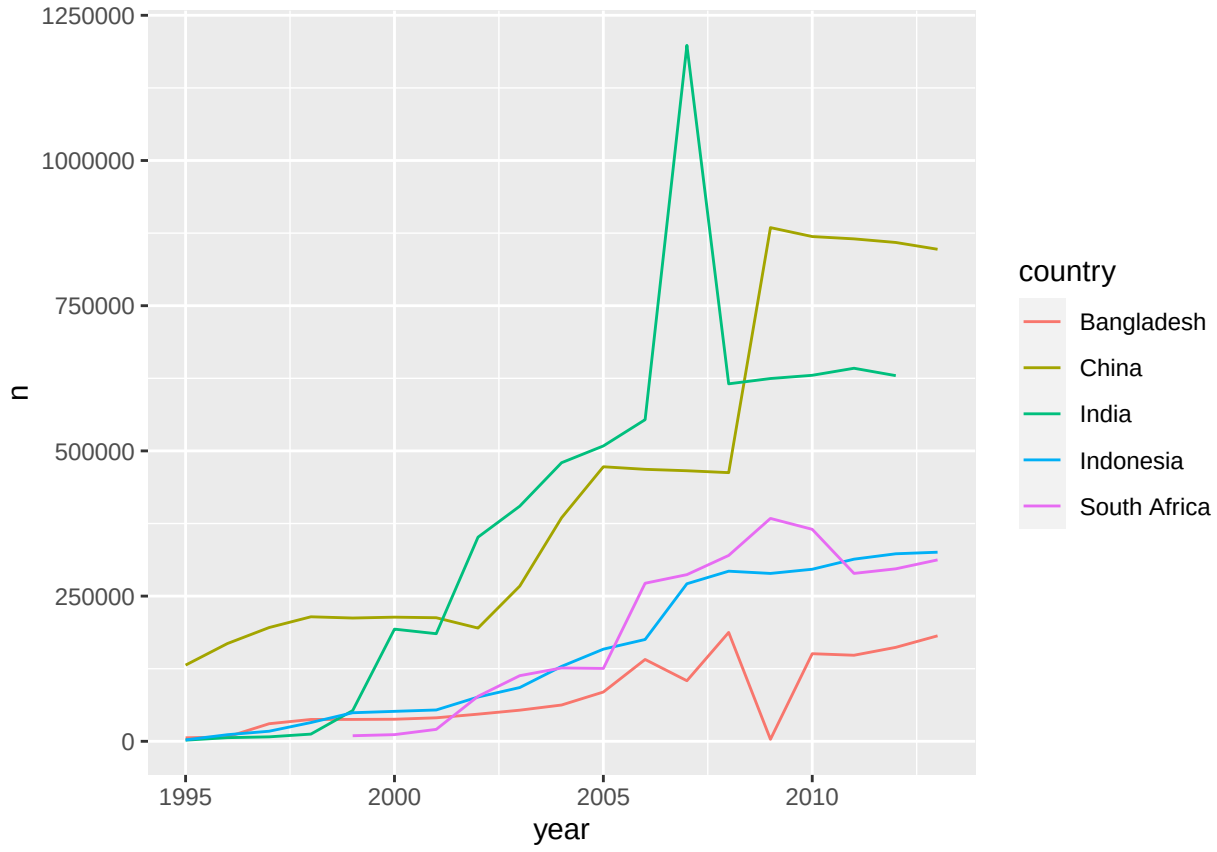
4. Producing these ratios by year (ignoring county) is probably a bad idea. Why?

Result: 1. Make a plot that address the main question (To what extent is tuberculosis associated with a specific sex and has this changed from 1997 onward?) Think carefully which kind of plot you are going to use, you want to uncover the general pattern but also learn specifics about your data. 1. Write a quick summary of lessons learned from your final data visualization. What is the general conclusion from this plot? Did you find any other variable information from your plot?

A small multiples plot faceting by country would probably be very difficult and unclear given the number of countries in the data. Focusing on those countries with the largest changes or absolute magnitudes of TB cases after providing the context above is another option.

```
#countries with the most cases of TB
who_tidy4 %>%
  group_by(country, year) %>%
  summarise(n=sum(cases)) %>%
  ungroup() %>%
  group_by(country) %>%
  mutate(total_country=sum(n)) %>%
  filter(total_country>1000000) %>%
  ggplot(aes(x=year,y=n,colour=country))+
  geom_line()
```

```
## `summarise()` has grouped output by 'country'. You can override using the `.groups` argument.
```



1.3 Unseen untidy data (15 pts)

1. The data set world_bank_pop is messy. Tidy it, show each of your steps and at the end write a short paragraph of what you just did.

```
new_worldbankpop <- world_bank_pop %>%
  gather(year, value, `2000`:`2017`, na.rm = TRUE) %>%
  pivot_wider(names_from = indicator, values_from = value) %>%
  arrange(country)

new_worldbankpop %>% as.tibble()
```

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
```

```
## # A tibble: 4,728 x 6
##    country year  SP.URB.TOTL SP.URB.GROW SP.POP.TOTL SP.POP.GROW
##    <chr>   <chr>       <dbl>       <dbl>       <dbl>       <dbl>
##  1 ABW     2000        42444      1.18         90853      2.06
##  2 ABW     2001        43048      1.41         92898      2.23
##  3 ABW     2002        43670      1.43         94992      2.23
##  4 ABW     2003        44246      1.31         97017      2.11
##  5 ABW     2004        44669      0.951        98737      1.76
```

```
##  6 ABW     2005        44889      0.491       100031      1.30
##  7 ABW     2006        44881     -0.0178      100832      0.798
##  8 ABW     2007        44686     -0.435       101220      0.384
##  9 ABW     2008        44375     -0.698       101353      0.131
## 10 ABW     2009        44052     -0.731       101453      0.0986
## # ... with 4,718 more rows
```

2 Data Types Strings (15 pts)

Hint: try lots of test cases to be sure you get it right 1. Write a regular expression to match any superhero name that ends with man and that is shorter or equal to 8 characters. This means it should be able to match Batman but not Spiderman (be careful I don't want it to match a regular man.) Prove your regular expression works with three examples:

```
words[grep(".........", words)]
```

```
##   [1] "absolute"    "advertise"   "afternoon"   "although"    "apparent"
##   [6] "approach"    "appropriate" "associate"   "authority"   "available"
##  [11] "brilliant"   "business"    "chairman"    "character"   "Christmas"
##  [16] "colleague"   "committee"   "community"   "complete"    "condition"
##  [21] "consider"    "continue"    "contract"    "converse"    "decision"
##  [26] "definite"    "department"  "describe"    "difference"  "difficult"
##  [31] "district"    "document"    "electric"    "encourage"   "environment"
##  [36] "especial"    "evidence"    "exercise"    "experience"  "function"
##  [41] "hospital"    "identify"    "important"   "increase"    "individual"
##  [46] "industry"    "interest"    "introduce"   "language"    "minister"
##  [51] "necessary"   "occasion"    "opportunity" "organize"    "original"
##  [56] "otherwise"   "paragraph"   "particular"  "photograph"  "position"
##  [61] "positive"    "possible"    "practise"    "pressure"    "previous"
##  [66] "probable"    "programme"   "question"    "recognize"   "recommend"
##  [71] "relation"    "remember"    "represent"   "research"    "resource"
##  [76] "responsible" "saturday"    "scotland"    "secretary"   "separate"
##  [81] "specific"    "standard"    "straight"    "strategy"    "structure"
##  [86] "surprise"    "telephone"   "television"  "terrible"    "therefore"
##  [91] "thirteen"    "thousand"    "thursday"    "together"    "tomorrow"
##  [96] "transport"   "understand"  "university"  "wednesday"   "yesterday"
```

```
paste0(".........", "man")
```

```
## [1] ".........man"
```

2. Given the corpus of fruits in stringer::fruit, create regular expressions that find all fruits that: 1.Ends with "t".

```
fruit[grep("t$", fruit)]
```

```
##  [1] "apricot"      "blackcurrant" "breadfruit"   "coconut"      "currant"
##  [6] "dragonfruit"  "eggplant"     "grapefruit"   "jackfruit"    "kiwi fruit"
## [11] "kumquat"      "loquat"       "nut"          "passionfruit" "redcurrant"
## [16] "star fruit"   "ugli fruit"
```

2. Starts with "h"

```
fruit[grep("^h", fruit)]
```

```
## [1] "honeydew"    "huckleberry"
```

3. Are exactly 6 letters long. (Don't use str_length()!)

```
fruit[grep("^......$", fruit)]
```

```
##  [1] "banana" "cherry" "damson" "durian" "feijoa" "jambul" "jujube" "loquat"
##  [9] "lychee" "orange" "pamelo" "papaya" "pomelo" "quince" "raisin"
```

4. Are 10 letters or longer. (Note: including all the output here would make grading difficult. Instead, use sum(str_detect(stringr::words,regex)) to count the number of strings that match each of the patterns above)

```
fruit[grep("..........", fruit)]
```

```
##  [1] "bell pepper"       "blackberry"        "blackcurrant"
##  [4] "blood orange"      "boysenberry"       "breadfruit"
##  [7] "canary melon"      "cantaloupe"        "chili pepper"
## [10] "clementine"        "cloudberry"        "dragonfruit"
## [13] "elderberry"        "goji berry"        "gooseberry"
## [16] "grapefruit"        "huckleberry"       "kiwi fruit"
## [19] "passionfruit"      "pomegranate"       "purple mangosteen"
## [22] "redcurrant"        "rock melon"        "salal berry"
## [25] "star fruit"        "strawberry"        "ugli fruit"
## [28] "watermelon"
```

3. Create regular expressions to find all words in stringr:words that meet the following criteria. In addition, please provide two test cases where your regular expression returns a match and two test cases that do not return a match.
4. Start with an a or an o.

```
words[grep("^a", words)]
```

```
##  [1] "a"           "able"        "about"       "absolute"    "accept"
##  [6] "account"     "achieve"     "across"      "act"         "active"
## [11] "actual"      "add"         "address"     "admit"       "advertise"
## [16] "affect"      "afford"      "after"       "afternoon"   "again"
## [21] "against"     "age"         "agent"       "ago"         "agree"
## [26] "air"         "all"         "allow"       "almost"      "along"
## [31] "already"     "alright"     "also"        "although"    "always"
## [36] "america"     "amount"      "and"         "another"     "answer"
## [41] "any"         "apart"       "apparent"    "appear"      "apply"
## [46] "appoint"     "approach"    "appropriate" "area"        "argue"
## [51] "arm"         "around"      "arrange"     "art"         "as"
## [56] "ask"         "associate"   "assume"      "at"          "attend"
## [61] "authority"   "available"   "aware"       "away"        "awful"
```

```
words[grep("^o", words)]
```

```
##  [1] "obvious"     "occasion"    "odd"         "of"          "off"
##  [6] "offer"       "office"      "often"       "okay"        "old"
## [11] "on"          "once"        "one"         "only"        "open"
## [16] "operate"     "opportunity" "oppose"      "or"          "order"
## [21] "organize"    "original"    "other"       "otherwise"   "ought"
## [26] "out"         "over"        "own"
```

2. That only contain contain consonants. (Hint: thinking about matching "not"- vowels)

```
str_subset(stringr::words, "[aeiou]", negate=TRUE)
```

```
## [1] "by"  "dry" "fly" "mrs" "try" "why"
```

3. That are berries, i.e., contain the word berry

```
fruit[grep("berry", fruit)]
```

```
##  [1] "bilberry"    "blackberry"  "blueberry"   "boysenberry" "cloudberry"
##  [6] "cranberry"   "elderberry"  "goji berry"  "gooseberry"  "huckleberry"
## [11] "mulberry"    "raspberry"   "salal berry" "strawberry"
```

4. End with ine or een.

```
fruit[grep("ine$", fruit)]
```

```
## [1] "clementine" "mandarine"  "nectarine"  "tangerine"
```

```
fruit[grep("een$", fruit)]
```

```
## [1] "purple mangosteen"
```

```
str_subset(stringr::words, "(ine|een)$")
```

```
## [1] "between"  "engine"   "fine"     "green"    "imagine"  "line"     "machine"
## [8] "nine"     "thirteen"
```

4. Show how telephone numbers are written in your country with three examples. Create a regular expression that will match telephone numbers are commonly ass written in your country.

In the United States, phone numbers have a format 123-456-7890 or (123)456-7890).

```
US_PHONE <- c("707-567-0315", "(707)567-0315", "(707) 567-0315", "1234-5678")
str_view(US_PHONE, "\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d")
```

```
707-567-0315
(707)567-0315
(707) 567-0315
1234-5678
```

```
str_view(US_PHONE,
         "[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]")
```

```
707-567-0315
(707)567-0315
(707) 567-0315
1234-5678
```

```
str_view(US_PHONE,
         "\\(\\d{3}\\)\\s*\\d{3}-\\d{4}")
```

```
707-567-0315
(707)567-0315
(707) 567-0315
1234-5678
```