Stat 260, Lecture 6, Tidy Data

David Stenning

Load packages and datasets

```
library(tidyverse)
table1 <- read_csv("lec06table1.csv",
                   col_types=cols(
                     country=col_character(),
                     year=col_integer(),
                     cases=col_integer(),
                     population=col_integer()
                   )
)
```

Create tibbles used in examples

```
table2 <- table1 %>%
  gather(cases,population,key="type",value="count") %>%
  arrange(country,year)
table3 <- table1 %>%
  mutate(rate = paste(cases,population,sep="/")) %>%
  select(-cases,-population)
table4a <- table1 %>%
  select(country,year,cases) %>%
  spread(key=year,value=cases)
table4b <- table1 %>%
  select(country,year,population) %>%
  spread(key=year,value=population)
```

Reading

Required Reading:

- Tidy Data: Chapter 12 of online text.

Useful References:

- Advanced: Wickham (2014) [https://www.jstatsoft.org/index.php/jss/article/view/v059i10/v59i10.pdf]
- Data import (readr/tidyr) cheatsheet at [https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf]

Tidy data

- In a tidy dataset,
  - each variable has its own column,
  - each observation has its own row, and
  - each value has its own cell.
- (See Figure 12.1 of the online text.)
- The tibble `table1` that we read in earlier is tidy:

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 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
```

Why is table1 tidy?

- It is never straightforward to answer this.
- These are WHO data on Tuberculosis cases.
  - The variables are country, year, number of cases and population
  - The observations are country/year combinations.
  - Each value is in its own cell.
- `country` and `year` describe the observational unit, and so there must be one row for each, and there must be variables that record `country` and `year`.
- `cases` and `population` are what we measure on the observational unit and so must be variables.

Non-tidy data

- There are many ways to be non-tidy.
- **Exercise:** Why are `table2` and `table4a` not tidy?

```
print(table2,n=6)
```

```
## # A tibble: 12 x 4
##   country      year type           count
##   <chr>       <int> <chr>          <int>
## 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
## # ... with 6 more rows
```

```
print(table4a,n=6)
```

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

Why tidy?

- Good statistics (exploratory, visualization, modelling) requires that we identify the observational unit.
- R is efficient at computing with vectors, so variables as column-vectors are efficient.
- Tidyverse tools require it; e.g.,

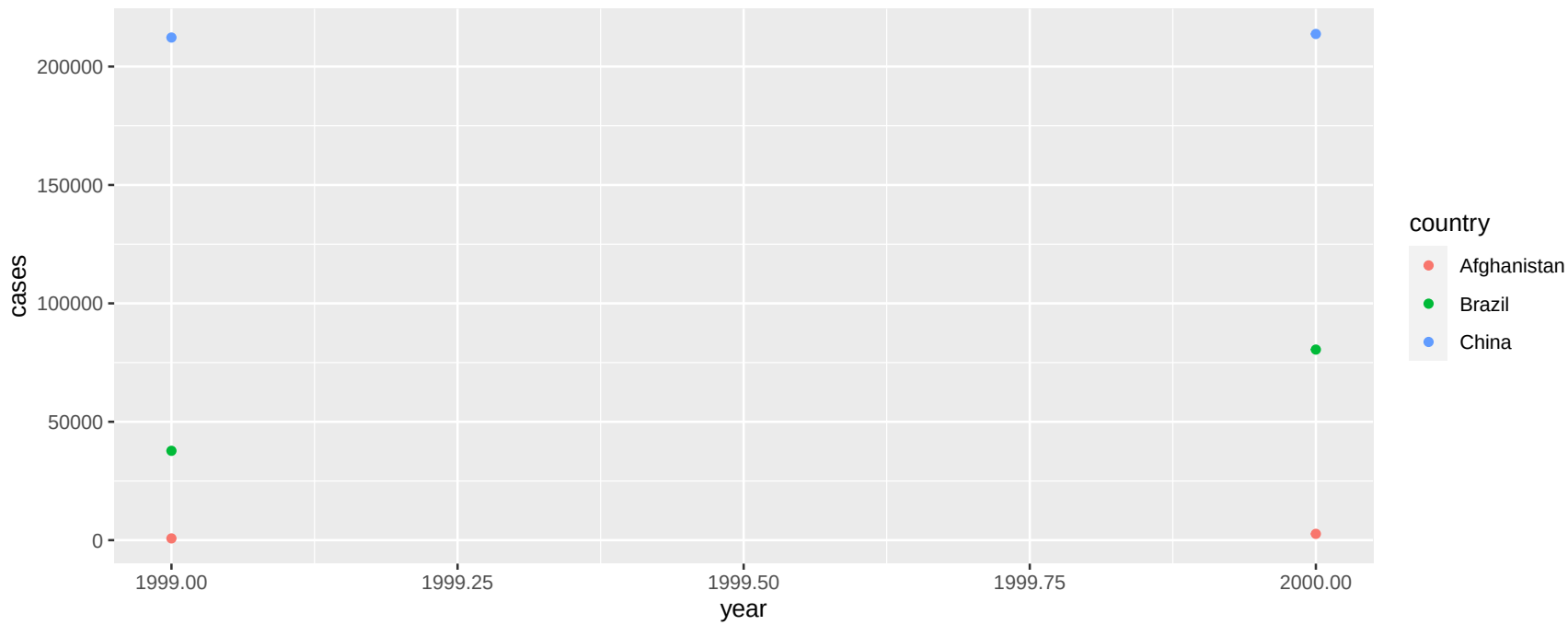
```
table1 %>% mutate(rate=cases/population*100000)
```

```
## # A tibble: 6 x 5
##   country      year  cases population  rate
##   <chr>       <int>  <int>      <int> <dbl>
## 1 Afghanistan  1999    745   19987071  3.73
## 2 Afghanistan  2000   2666   20595360 12.9
## 3 Brazil       1999  37737  172006362 21.9
## 4 Brazil       2000  80488  174504898 46.1
## 5 China        1999 212258 1272915272 16.7
## 6 China        2000 213766 1280428583 16.7
```

```r
table1 %>% group_by(year) %>% summarize(sum(cases))
```

```
## # A tibble: 2 x 2
##    year `sum(cases)`
##   <int>        <int>
## 1  1999       250740
## 2  2000       296920
```

```r
ggplot(table1,aes(x=year,y=cases,color=country)) + geom_point()
```



Exercises

- Compute `rate` from `table2`.
- Compute `rate` from `table4a` and `table4b`.

Gathering

- The problem with table 4a is that the cases variable is split across the columns 1999 and 2000 that must be "gathered" into one.
- Each value of cases needs a label (key) to tell us which year it comes from.

```
table4a %>% gather(`1999`,`2000`,key=year,value=cases)
```

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

- **Exercise** Repeat for `table4b`

Gathering: another example

Billboard Top 100 rankings of songs

```
bb <- read_csv("billboard.csv")
bb
```

```
## # A tibble: 317 x 83
##     year artist.inverted track time  genre date.entered date.peaked x1st.wee
##    <dbl> <chr>           <chr> <tim> <chr> <date>       <date>           <dbl
##  1  2000 Destiny's Child Inde~ 03:38 Rock  2000-09-23   2000-11-18         7
##  2  2000 Santana         Mari~ 04:18 Rock  2000-02-12   2000-04-08         1
##  3  2000 Savage Garden   I Kn~ 04:07 Rock  1999-10-23   2000-01-29         7
##  4  2000 Madonna         Music 03:45 Rock  2000-08-12   2000-09-16         4
##  5  2000 Aguilera, Chri~ Come~ 03:38 Rock  2000-08-05   2000-10-14         5
##  6  2000 Janet           Does~ 04:17 Rock  2000-06-17   2000-08-26         5
##  7  2000 Destiny's Child Say ~ 04:31 Rock  1999-12-25   2000-03-18         8
##  8  2000 Iglesias, Enri~ Be W~ 03:36 Latin 2000-04-01   2000-06-24         6
##  9  2000 Sisqo           Inco~ 03:52 Rock  2000-06-24   2000-08-12         7
## 10  2000 Lonestar        Amaz~ 04:25 Coun~ 1999-06-05   2000-03-04         8
## # ... with 307 more rows, and 75 more variables: x2nd.week <dbl>,
## #   x3rd.week <dbl>, x4th.week <dbl>, x5th.week <dbl>, x6th.week <dbl>,
## #   x7th.week <dbl>, x8th.week <dbl>, x9th.week <dbl>, x10th.week <dbl>,
## #   x11th.week <dbl>, x12th.week <dbl>, x13th.week <dbl>, x14th.week <dbl>,
## #   x15th.week <dbl>, x16th.week <dbl>, x17th.week <dbl>, x18th.week <dbl>,
## #   x19th.week <dbl>, x20th.week <dbl>, x21st.week <dbl>, x22nd.week <dbl>,
## #   x23rd.week <dbl>, x24th.week <dbl>, x25th.week <dbl>, x26th.week <dbl>,
## #   x27th.week <dbl>, x28th.week <dbl>, x29th.week <dbl>, x30th.week <dbl>,
## #   ...
```

Structure of the Billboard table

- Columns `year` through `date.peaked` describe the song, then `x1st.week` through `x76th.week` are the chart positions for the first through 76th weeks.
  - If a song is on the chart for less than 76 weeks, its position is `NA` for any missing weeks.
- Weeks are not variables, they are the time data for the time series.

Tidying the Billboard data

- Main step is to gather the rankings in the different weeks into a `rank` variable.
- Before gathering, will select/rename some of the variables.
- After gathering, will create some new variables and sort the data frame.

Select and rename

- Won't need `time` or `genre`.
  - Recall that `select()` from `dplyr` can use `-` to de-select
- Rename `artist.inverted`
  - Recall that `rename()` from `dplyr` takes arguments of the form `newname = oldname`

```
bb <-
  bb %>% select(-time,-genre) %>%
  rename(artist = artist.inverted)
```

Gather the weeks into a "long" version of the Billboard data

- Leave each song info variable as-is.
- The "values", are the chart positions.
- The weeks are the "keys" for these values.
- We want to create key-value pairs for each observation.
  - There will be missing values, which we can remove.

```r
bb %>% gather(x1st.week:x76th.week,key=week,value=rank,na.rm=TRUE) %>%
  mutate(week= parse_number(week)) %>% # replace, e.g., x1st.week with 1, ...
  arrange(artist,track,week)
```

```
## # A tibble: 5,307 x 7
##     year artist  track                    date.entered date.peaked  week
##    <dbl> <chr>   <chr>                    <date>       <date>      <dbl> <
##  1  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      1
##  2  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      2
##  3  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      3
##  4  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      4
##  5  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      5
##  6  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      6
##  7  2000 2 Pac   Baby Don't Cry (Keep Ya H~ 2000-02-26   2000-03-11      7
##  8  2000 2Ge+her The Hardest Part Of Break~ 2000-09-02   2000-09-09      1
##  9  2000 2Ge+her The Hardest Part Of Break~ 2000-09-02   2000-09-09      2
## 10  2000 2Ge+her The Hardest Part Of Break~ 2000-09-02   2000-09-09      3
## # ... with 5,297 more rows
```

Spreading

- Whereas gathering makes a wide dataset long, spreading makes a long dataset wide.
- Spread when observations are split across multiple rows.
- E.G., `table2` has observations for each country/year split across two rows:

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  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
```

Spreading `table2`

- The column `type` contains the keys for the values in `count` that belong in columns `cases` and `population`, respectively.

```
table2 %>% spread(key=type,value=count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 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
```

- **Exercise** Select `country`, `year` and `cases` from `table1` and use spread to obtain a table with rows for each year and columns for each country. (Note: such data is **not** tidy.)

Separating

- `separate()` splits a column on a specified separator, or at a specified character number.

```
print(table3,n=4)
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## # ... with 2 more rows
```

```
table3 %>% separate(rate,into=c("cases","population"),sep="/") %>% print(n=4)
```

```
## # A tibble: 6 x 4
##   country      year cases population
##   <chr>       <int> <chr> <chr>
## 1 Afghanistan  1999 745   19987071
## 2 Afghanistan  2000 2666  20595360
## 3 Brazil       1999 37737 172006362
## 4 Brazil       2000 80488 174504898
## # ... with 2 more rows
```

- Notice that `cases` and `population` are character columns.

Separating based on character number

```
print(table3,n=4)
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## # ... with 2 more rows
```

```
table3 %>% separate(rate,into=c("first","remainder"),sep=1)
```

```
## # A tibble: 6 x 4
##   country      year first remainder
##   <chr>       <int> <chr> <chr>
## 1 Afghanistan  1999 7     45/19987071
## 2 Afghanistan  2000 2     666/20595360
## 3 Brazil       1999 3     7737/172006362
## 4 Brazil       2000 8     0488/174504898
## 5 China        1999 2     12258/1272915272
## 6 China        2000 2     13766/1280428583
```

Convert type of columns after separating

```
table3 %>% separate(rate,into=c("cases","population"),sep="/",
                    convert=TRUE)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 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
```

Missing data

- When we used `gather()` on the Billboard data we set `na.rm=TRUE` to remove weeks where a given track was not on the charts:

```
bb %>% select(track,x23rd.week:x25th.week) %>% print(n=4)
```

```
## # A tibble: 317 x 4
##   track                   x23rd.week x24th.week x25th.week
##   <chr>                        <dbl>      <dbl>      <dbl>
## 1 Independent Women Part I        10         12         15
## 2 Maria, Maria                    26         36         48
## 3 I Knew I Loved You               8         12         14
## 4 Music                           29         44         NA
## # ... with 313 more rows
```

- The missings in bb are "explicit"; when `gather()` removes them they become "implicit" (e.g., no row for week 25 for track 4).

Making implicit missing data explicit

- `spread()` will make implicit missing values explicit if needed for a row.

```
stocks <- tibble( year=c(2015,2016,2016), qtr = c(1,1,2),
                  return = c(1.0,2.0,3.0))
stocks
```

```
## # A tibble: 3 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1      1
## 2  2016     1      2
## 3  2016     2      3
```

```
stocks %>% spread(key=year,value=return)
```

```
## # A tibble: 2 x 3
##     qtr `2015` `2016`
##   <dbl>  <dbl>  <dbl>
## 1     1      1      2
## 2     2     NA      3
```

Make implicit explicit with `complete()`

- `complete()` creates rows for all combinations of input variables and fills in missing values where necessary.

```
stocks %>% complete(year,qtr)
```

```
## # A tibble: 4 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1      1
## 2  2015     2     NA
## 3  2016     1      2
## 4  2016     2      3
```

Case Study: WHO TB data

- The `who` dataset comes with `tidyr`. We'll use a related (less tidy) version from the WHO website.

```
tb <- read_csv("tb.csv")
```

Structure of TB table

- First column is 2-letter country code, second is year, third is number of new cases for that country/year.
- Then come TB counts for different gender/age categories.
  - `new_sp` is "new cases by positive pulmonary smear assay"
  - gender is `m` or `f`
  - two special age categories 0-4, 5-14,
  - age categories 0-14, 15-24, 25-34, 35-44, 45-54, 55-65, 65+, unknown (`u`)
- Gender/age columns are not variables, they are data on the observed units.
- Tidy data would have one row for each country, year, gender and age category, with a column of counts

Tidying the TB data

- Recall structure of the data: country, year, count of new cases, counts of new cases by gender/age categories.

```
names(tb)[1:10]
```

```
##  [1] "iso2"         "year"         "new_sp"       "new_sp_m04"   "new_sp_m51
##  [6] "new_sp_m014"  "new_sp_m1524" "new_sp_m2534" "new_sp_m3544" "new_sp_m45
```

- Main step is to "gather" TB prevalence in the different gender/age categories into a count variable.
  - Complicated by the coding of gender/age categories
- Before gathering, will remove unneeded variables.

Remove variables

- Won't need overall count
- Special categories 0-4 and 5-14 overlap with 0-14, so remove
- Age unknown not useful for analysing trends, so remove

```
tb <- select(tb,-new_sp, -contains("04"), -contains("514"),
             -new_sp_mu, -new_sp_fu)
tb
```

```
## # A tibble: 5,769 x 16
##    iso2   year new_sp_m014 new_sp_m1524 new_sp_m2534 new_sp_m3544 new_sp_m45
##    <chr> <dbl>       <dbl>        <dbl>        <dbl>        <dbl>        <db
##  1 AD     1989          NA           NA           NA           NA
##  2 AD     1990          NA           NA           NA           NA
##  3 AD     1991          NA           NA           NA           NA
##  4 AD     1992          NA           NA           NA           NA
##  5 AD     1993          NA           NA           NA           NA
##  6 AD     1994          NA           NA           NA           NA
##  7 AD     1996           0            0            0            4
##  8 AD     1997           0            0            1            2
##  9 AD     1998           0            0            0            1
## 10 AD     1999           0            0            0            1
## # ... with 5,759 more rows, and 9 more variables: 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>
```

Gather counts for demographic groups

- Create demographic variable `demog` and count variable `count` by gathering over all variables except `iso2` and `year`.

```
tblong <- tb %>%
  gather(new_sp_m014:new_sp_f65,key=demog,value=count,na.rm=TRUE)
tblong
```

```
## # A tibble: 33,615 x 4
##    iso2   year demog        count
##    <chr> <dbl> <chr>        <dbl>
##  1 AD     1996 new_sp_m014     0
##  2 AD     1997 new_sp_m014     0
##  3 AD     1998 new_sp_m014     0
##  4 AD     1999 new_sp_m014     0
##  5 AD     2000 new_sp_m014     0
##  6 AD     2001 new_sp_m014     0
##  7 AD     2002 new_sp_m014     0
##  8 AD     2003 new_sp_m014     0
##  9 AD     2004 new_sp_m014     0
## 10 AD     2005 new_sp_m014     0
## # ... with 33,605 more rows
```

Separate gender from age category.

- First remove `new_sp_`, then separate result on first column

```
maxlen <- max(nchar(tblong$demog))
tb <- tblong %>% mutate(demog = substr(demog,8,maxlen)) %>%
  separate(demog, into=c("gender","agecat"),sep=1)
tb
```

```
## # A tibble: 33,615 x 5
##    iso2   year gender agecat count
##    <chr> <dbl> <chr>  <chr>  <dbl>
##  1 AD     1996 m      014        0
##  2 AD     1997 m      014        0
##  3 AD     1998 m      014        0
##  4 AD     1999 m      014        0
##  5 AD     2000 m      014        0
##  6 AD     2001 m      014        0
##  7 AD     2002 m      014        0
##  8 AD     2003 m      014        0
##  9 AD     2004 m      014        0
## 10 AD     2005 m      014        0
## # ... with 33,605 more rows
```