Stat 33B - Lecture 6

February 26, 2020

Announcements

Quiz 2 this Friday:

- Similar length to Quiz 1
- Multiple-choice and fill-in-the-blank questions.
- Covers everything up to the end of last week.

Review

Previous lecture:

- Tidy data
- The `pivot_wider` function

We took this untidy data:

```
library(tidyr)

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
```

And used the `pivot_wider()` function to rotate rows into columns:

```
pivot_wider(table2, names_from = type, values_from = 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
```

Rotating Columns into Rows

Some data sets put measurements for multiple observations in a single row.

In this table, each row contains the measurements for two observations of the `cases` variable:

```
table4a
```

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

To make this data tidy:

- Rotate the `1999` and `2000` columns into rows.
- New columns are `year` and `cases`.

Tidying this data set makes it longer.

Important Note: Earlier we saw negative indexing with positions:

```
table4a[-1]
```

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

Tidyverse functions (but not base R!) also support negative indexing with names!

Use `pivot_longer()` to rotate columns into rows:

```
pivot_longer(table4a, -country, 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
```

Need to specify:

- Columns to rotate as `cols`.
- Name(s) of new identifier column(s) as `names_to`.
- Name(s) of new measuerment column(s) as `values_to`.

See `vignette("pivot")` for more examples of using `tidyr`.

If you wanted to do this without `tidyr`:

1. Subset columns to separate `1999` and `2000` into two data frames.

2. Add a `year` column to each.
3. Rename the `1999` and `2000` columns to `cases`.
4. Stack the two data frames with `rbind()`.

```
col99 = table4a[-3]
col00 = table4a[-2]

names(col99)[2] = "cases"
col99["year"] = 1999

col99
```

```
## # A tibble: 3 x 3
##   country      cases  year
##   <chr>        <int> <dbl>
## 1 Afghanistan    745  1999
## 2 Brazil       37737  1999
## 3 China       212258  1999
```

```
names(col00)[2] = "cases"
col00["year"] = 2000

col00
```

```
## # A tibble: 3 x 3
##   country      cases  year
##   <chr>        <int> <dbl>
## 1 Afghanistan   2666  2000
## 2 Brazil       80488  2000
## 3 China       213766  2000
```

```
rbind(col99, col00)
```

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

Merging Data

A merge or “join” combines data from two separate data frames, based on some identifying values they have in common.

Recall the `pivot_wider()` example with `table2`.

To make `table2` tidy without `tidyr`:

1. Subset rows to separate `cases` and `population` values.
2. Remove the `type` column from each.
3. Rename the `count` column to `cases` and `population`.
4. Merge these two subsets by matching `country` and `year`.

The two subsets are:

```
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
```

```
cases = table2[table2$type == "cases", ]
pop = table2[table2$type == "population", ]

names(pop)[4] = "population"
pop = pop[-3]

pop
```

```
## # A tibble: 6 x 3
##   country      year population
##   <chr>       <int>      <int>
## 1 Afghanistan  1999   19987071
## 2 Afghanistan  2000   20595360
## 3 Brazil       1999  172006362
## 4 Brazil       2000  174504898
## 5 China        1999 1272915272
## 6 China        2000 1280428583
```

```
names(cases)[4] = "cases"
cases = cases[-3]
```

BAD ways to combine rows:

```
cases$population = pop$population

# OR

cbind(cases, pop["population"])
```

Dangerous if you're not 110% sure the order of rows matches!

Imagine if the rows were in a different order:

```
pop = pop[c(4, 5, 1, 2, 3, 6), ]
```

Then adding a new column or using `cbind()` would mix up the observations.

Instead, match rows on `country` and `year` with the `merge()` function:

```r
merge(cases, pop)
```

```
##       country year  cases population
## 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
```

The function automatically uses columns that have the same name in both tables to match.

More Merges

Data split across multiple tables are called *relational data*.

A column shared by several tables is called a *key*.

For example, a grocery store's inventory system might have:

- A table that lists stores
- A table that lists items (fruits, vegetables, etc)
- A table that lists quantity of each item at each store

See the bCourse for the data set:

```r
stores = readRDS("data/grocery/stores.rds")

stores
```

```
##   store_id   manager status          city
## 1        1      Chen     20       Oakland
## 2        2 Hernandez     10 San Francisco
## 3        3     Smith     30 San Francisco
## 4        4       Ali     20       Oakland
## 5        5     Rossi     30      Berkeley
```

```r
items = readRDS("data/grocery/items.rds")

items
```

```
##   ItemID ItemName       Variety Price    Source
## 1      1    Lemon        Eureka  0.50 San Diego
## 2      2   Orange      Valencia  1.00 San Diego
## 3      3    Apple Red Delicious  0.90   Spokane
## 4      4    Apple          Fuji  0.90   Spokane
## 5      5     Lime           Key  1.50  Veracruz
## 6      6    Mango       Alfonso  2.25   Lucknow
```

```r
inv = readRDS("data/grocery/inventory.rds")

inv
```

```
##    StoreID ItemID Qty
## 1        1      1 300
## 2        1      2 200
## 3        1      3 400
## 4        1      4 200
## 5        1      5 100
## 6        1      6 100
## 7        2      1 300
## 8        2      2 400
## 9        3      2 200
## 10       4      2 200
## 11       4      4 300
## 12       4      5 400
```

By default, the merge() function only keeps rows that match:

```
merge(stores, inv)
```

```
##    store_id   manager status          city StoreID ItemID Qty
## 1         1      Chen     20       Oakland       1      1 300
## 2         2 Hernandez     10 San Francisco       1      1 300
## 3         3     Smith     30 San Francisco       1      1 300
## 4         4       Ali     20       Oakland       1      1 300
## 5         5     Rossi     30      Berkeley       1      1 300
## 6         1      Chen     20       Oakland       1      2 200
## 7         2 Hernandez     10 San Francisco       1      2 200
## 8         3     Smith     30 San Francisco       1      2 200
## 9         4       Ali     20       Oakland       1      2 200
## 10        5     Rossi     30      Berkeley       1      2 200
## 11        1      Chen     20       Oakland       1      3 400
## 12        2 Hernandez     10 San Francisco       1      3 400
## 13        3     Smith     30 San Francisco       1      3 400
## 14        4       Ali     20       Oakland       1      3 400
## 15        5     Rossi     30      Berkeley       1      3 400
## 16        1      Chen     20       Oakland       1      4 200
## 17        2 Hernandez     10 San Francisco       1      4 200
## 18        3     Smith     30 San Francisco       1      4 200
## 19        4       Ali     20       Oakland       1      4 200
## 20        5     Rossi     30      Berkeley       1      4 200
## 21        1      Chen     20       Oakland       1      5 100
## 22        2 Hernandez     10 San Francisco       1      5 100
## 23        3     Smith     30 San Francisco       1      5 100
## 24        4       Ali     20       Oakland       1      5 100
## 25        5     Rossi     30      Berkeley       1      5 100
## 26        1      Chen     20       Oakland       1      6 100
## 27        2 Hernandez     10 San Francisco       1      6 100
## 28        3     Smith     30 San Francisco       1      6 100
## 29        4       Ali     20       Oakland       1      6 100
## 30        5     Rossi     30      Berkeley       1      6 100
## 31        1      Chen     20       Oakland       2      1 300
## 32        2 Hernandez     10 San Francisco       2      1 300
## 33        3     Smith     30 San Francisco       2      1 300
## 34        4       Ali     20       Oakland       2      1 300
## 35        5     Rossi     30      Berkeley       2      1 300
## 36        1      Chen     20       Oakland       2      2 400
## 37        2 Hernandez     10 San Francisco       2      2 400
## 38        3     Smith     30 San Francisco       2      2 400
## 39        4       Ali     20       Oakland       2      2 400
## 40        5     Rossi     30      Berkeley       2      2 400
## 41        1      Chen     20       Oakland       3      2 200
## 42        2 Hernandez     10 San Francisco       3      2 200
## 43        3     Smith     30 San Francisco       3      2 200
## 44        4       Ali     20       Oakland       3      2 200
## 45        5     Rossi     30      Berkeley       3      2 200
## 46        1      Chen     20       Oakland       4      2 200
## 47        2 Hernandez     10 San Francisco       4      2 200
## 48        3     Smith     30 San Francisco       4      2 200
## 49        4       Ali     20       Oakland       4      2 200
## 50        5     Rossi     30      Berkeley       4      2 200
## 51        1      Chen     20       Oakland       4      4 300
## 52        2 Hernandez     10 San Francisco       4      4 300
## 53        3     Smith     30 San Francisco       4      4 300
## 54        4       Ali     20       Oakland       4      4 300
## 55        5     Rossi     30      Berkeley       4      4 300
## 56        1      Chen     20       Oakland       4      5 400
## 57        2 Hernandez     10 San Francisco       4      5 400
## 58        3     Smith     30 San Francisco       4      5 400
## 59        4       Ali     20       Oakland       4      5 400
## 60        5     Rossi     30      Berkeley       4      5 400
```

Use the `all` parameter to force all rows from both tables to show up:

```
merge(stores, inv, by.x = "store_id", by.y = "StoreID", all = TRUE)
```

```
##    store_id   manager status          city ItemID Qty
## 1         1      Chen     20       Oakland      1 300
## 2         1      Chen     20       Oakland      2 200
## 3         1      Chen     20       Oakland      3 400
## 4         1      Chen     20       Oakland      4 200
## 5         1      Chen     20       Oakland      5 100
## 6         1      Chen     20       Oakland      6 100
## 7         2 Hernandez     10 San Francisco      1 300
## 8         2 Hernandez     10 San Francisco      2 400
## 9         3     Smith     30 San Francisco      2 200
## 10        4       Ali     20       Oakland      2 200
## 11        4       Ali     20       Oakland      4 300
## 12        4       Ali     20       Oakland      5 400
## 13        5     Rossi     30      Berkeley     NA  NA
```

There are also `all.x` and `all.y` parameters to only force all rows from one table.

Use the `by` parameter to specify the key. If the name of the key is different for the two tables, use `by.x` and `by.y`.

```
merge(stores, inv, by.x = "store_id", by.y = "StoreID")
```

```
##    store_id   manager status          city ItemID Qty
## 1         1      Chen     20       Oakland      1 300
## 2         1      Chen     20       Oakland      2 200
## 3         1      Chen     20       Oakland      3 400
## 4         1      Chen     20       Oakland      4 200
## 5         1      Chen     20       Oakland      5 100
## 6         1      Chen     20       Oakland      6 100
## 7         2 Hernandez     10 San Francisco      1 300
## 8         2 Hernandez     10 San Francisco      2 400
## 9         3     Smith     30 San Francisco      2 200
## 10        4       Ali     20       Oakland      2 200
## 11        4       Ali     20       Oakland      4 300
## 12        4       Ali     20       Oakland      5 400
```

If you're familiar with SQL `JOIN`s, the `merge()` function is the same idea. Specifically:

- `INNER JOIN` is `all = FALSE` (the default)
- `LEFT JOIN` is `all.x = TRUE`
- `RIGHT JOIN` is `all.y = TRUE`
- `OUTER JOIN` is `all = TRUE`