Getting / cleaning data 2

Final group project

Final group project

- Group size: Three or four students
- If you'd like, you may form your own groups. For any students who do not form a group, I will randomly assign groups (or add on to groups that have started).

Final group project

Important dates:

- October 17: Due date for creating groups. Email me your group members.
- October 24: Due date (by start of class) for a two-paragraph summary of the question you'd like to answer, including some ideas on where you might find the data.
- December 5: First submission of written report will be due.
- Week of December 12: Final presentation and final draft of written report due.

Final group project

- You will have in-class group work time during weeks 10–12 to work on this. This will also require some work with your group outside of class.
- You will be able to get feedback and help from me during the in-class group work time.
- Your project should not use any datasets from your own research or from other classes.
- Part of the grade will be on the writing and presentation of the final project.

Final group project

To get an idea of what your final product should look like, check out these links:

- Does Christmas come earlier each year?
- Hilary: the most poisoned baby name in US history
- Every Guest Jon Stewart Ever Had On "The Daily Show"
- Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past?
- Billion-Dollar Billy Beane

Final group project

Part of your final project will be to design a Shiny app.

To see some examples of Shiny apps, see the Shiny gallery.

Joining datasets

Joining datasets

So far, you have only worked with a single data source at a time. When you work on your own projects, however, you typically will need to merge together two or more datasets to create the a data frame to answer your research question.

For example, for air pollution epidemiology, you will often have to join several datasets:

- Health outcome data (e.g., number of deaths per day)
- Air pollution concentrations
- Weather measurements (since weather can be a confounder)
- Demographic data

*_join functions

The dplyr package has a family of different functions to join two dataframes together, the *_join family of functions. All combine two dataframes, which I'll call x and y here.

The functions include:

- inner_join(x, y): Keep only rows where there are observations in both x and y.
- left_join(x, y): Keep all rows from x, whether they have a match in y or not.
- right_join(x, y): Keep all rows from y, whether they have a match in x or not.
- full_join(x, y): Keep all rows from both x and y, whether they have a match in the other dataset or not.

*_join functions

In the examples, I'll use two datasets, x and y. Both datasets include the column course. The other column in x is grade, while the other column in y is day. Observations exist for courses x and y in both datasets, but for w and z in only one dataset.

```
x <- data.frame(course = c("x", "y", "z"),
                grade = c(90, 82, 78))
y <- data.frame(course = c("w", "x", "y"),
                day = c("Tues", "Mon / Fri", "Tue"))
```

*_join functions

Here is what these two example datasets look like:

```
x
```

```
##   course grade
## 1      x    90
## 2      y    82
## 3      z    78
```

```
y
```

```
##   course         day
## 1      w        Tues
## 2      x Mon / Fri
## 3      y         Tue
```

*_join functions

With `inner_join`, you'll only get the observations that show up in both datasets. That means you'll lose data on `z` (only in the first dataset) and `w` (only in the second dataset).

```
inner_join(x, y)
```

```
## Joining, by = "course"
```

```
##   course grade       day
## 1      x    90 Mon / Fri
## 2      y    82       Tue
```

*_JOIN FUNCTIONS

With `left_join`, you'll keep everything in `x` (the "left" dataset), but not keep things in `y` that don't match something in `x`. That means that, here, you'll lose `w`:

```
left_join(x, y)
```

```
## Joining, by = "course"
```

```
##   course grade       day
## 1      x    90 Mon / Fri
## 2      y    82       Tue
## 3      z    78      <NA>
```

*_join functions

right_join is the opposite:

```
right_join(x, y)
```

```
## Joining, by = "course"
```

```
##   course grade        day
## 1      w    NA       Tues
## 2      x    90 Mon / Fri
## 3      y    82        Tue
```

*_JOIN FUNCTIONS

full_join keeps everything from both datasets:

```
full_join(x, y)
```

```
## Joining, by = "course"
```

```
##   course grade        day
## 1      x    90 Mon / Fri
## 2      y    82       Tue
## 3      z    78      <NA>
## 4      w    NA      Tues
```

Tidy data

Tidy data

All of the material in this section comes directly from Hadley Wickham's paper on tidy data. You will need to read this paper to prepare for the quiz on this section.

Characteristics of tidy data

Characteristics of tidy data are:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Getting your data into a "tidy" format makes it easier to model and plot. By taking the time to tidy your data at the start of an analysis, you will save yourself time, and make it easier to plan out, later steps.

Five common problems

Here are five common problems that Hadley Wickham has identified that keep data from being tidy:

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
3. Variables are stored in both rows and columns.
4. Multiple types of observational units are stored in the same table.
5. A single observational unit is stored in multiple tables.

In the following slides, I'll give examples of each of these problems.

FIVE COMMON PROBLEMS

(1.) Column headers are values, not variable names.

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don't know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah's Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

FIGURE 1:

Five common problems

Solution:

religion	income	freq
Agnostic	<$10k	27
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$75-100k	122
Agnostic	$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

Five common problems

(2.) Multiple variables are stored in one column.

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

Five common problems

Solution:

country	year	sex	age	cases
AD	2000	m	0-14	0
AD	2000	m	15-24	0
AD	2000	m	25-34	1
AD	2000	m	35-44	0
AD	2000	m	45-54	0
AD	2000	m	55-64	0
AD	2000	m	65+	0
AE	2000	m	0-14	2
AE	2000	m	15-24	4
AE	2000	m	25-34	4
AE	2000	m	35-44	6
AE	2000	m	45-54	5
AE	2000	m	55-64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

Five common problems

(3.) Variables are stored in both rows and columns.

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Figure 5:

Five common problems

Solution:

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

Figure 6:

FIVE COMMON PROBLEMS

(4.) Multiple types of observational units are stored in the same table.

year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

FIGURE 7:

Five common problems

Solution:

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

Figure 8:

Five common problems

(5.) A single observational unit is stored in multiple tables.

Example: exposure and outcome data stored in different files:

- File 1: Daily mortality counts
- File 2: Daily air pollution measurements

gather / spread

There are two functions from the tidyr package (another member of the tidyverse) that you can use to change between wide and long data: gather() and spread(). (Example from tidyr help files.)

Here is some wide data:

```
wide_stocks[1:3, ]
```

```
##          time         X          Y          Z
## 1 2009-01-01 0.0680520 -0.6403972 -1.7782055
## 2 2009-01-02 0.1118558  1.5578847  1.3729563
## 3 2009-01-03 0.6887925 -1.8477021 -0.2596041
```

gather / spread

Use gather to convert wide data to long data:

```
long_stocks <- gather(wide_stocks, stock, price, -time)
long_stocks[1:5, ]
```

```
##         time stock      price
## 1 2009-01-01     X  0.0680520
## 2 2009-01-02     X  0.1118558
## 3 2009-01-03     X  0.6887925
## 4 2009-01-04     X -0.3080889
## 5 2009-01-05     X -2.0531876
```

Gather / Spread

Use spread to convert long data to wide data:

```
stocks <- spread(long_stocks, stock, price)
stocks[1:5, ]
```

```
##         time          X          Y          Z
## 1 2009-01-01  0.0680520 -0.6403972 -1.7782055
## 2 2009-01-02  0.1118558  1.5578847  1.3729563
## 3 2009-01-03  0.6887925 -1.8477021 -0.2596041
## 4 2009-01-04 -0.3080889  1.8264767 -2.7560475
## 5 2009-01-05 -2.0531876  3.0512822  2.8554204
```

Tidying with dplyr

Tidy `nepali` data

Say we wanted to tidy up the data by:

- Move variables that are constant for each child across all measurements (e.g., `mage`, `lit`, `died`, `alive`) to another dataset
- Determine each child's age at first measurement
- Limit the measurement dataset to just males
- Add new variables for (1) height-to-weight ratio and (2) months since first measurement

Tidy nepali data

Move variables that are constant for each child across all measurements (e.g., `mage`, `lit`, `died`, `alive`) to another dataset:

```
child_constants <- select(nepali, id, mage, lit, died, alive) %>%
        group_by(id) %>%
        summarize(mage = mean(mage), lit = mean(lit),
                  died = mean(died), alive = mean(alive))
child_constants[1:2, ]
```

```
## # A tibble: 2 × 5
##       id  mage   lit  died alive
##    <int> <dbl> <dbl> <dbl> <dbl>
## 1 120011    35     0     2     5
## 2 120012    35     0     2     5
```

Tidy nepali data

Determine each child's age at first measurement:

```
first_age <- group_by(nepali, id) %>%
        summarize(first_age = min(age))
first_age[1:2, ]
```

```
## # A tibble: 2 × 2
##        id first_age
##     <int>     <int>
## 1 120011        41
## 2 120012        57
```

Tidy nepali data

- Limit the measurement dataset with just males
- Add new variables for (1) height-to-weight ratio and (2) months since first measurement

```
child_measures <- select(nepali, -mage, -lit, -died, -alive) %>%
        filter(sex == 1) %>%
        left_join(first_age, by = "id") %>%
        mutate(ht_wt_ratio = ht / wt,
               months = age - first_age)

child_measures[1:2, ]
```

```
##        id sex   wt   ht age first_age ht_wt_ratio months
## 1 120011   1 12.8 91.2  41        41    7.125000      0
## 2 120011   1 12.8 93.9  45        41    7.335938      4
```

More with dplyr

DPLYR

So far, you've used several `dplyr` functions:

- `rename`
- `filter`
- `select`
- `mutate`
- `group_by`
- `summarize`

Some other useful `dplyr` functions to add to your toolbox are:

- `arrange` (including with `desc`)
- `slice`
- `mutate` with `group_by`

ARRANGE

Re-order data:

```
nepali[1:2, ]
```

```
##       id sex   wt   ht mage lit died alive age
## 1 120011   1 12.8 91.2   35   0    2     5  41
## 2 120011   1 12.8 93.9   35   0    2     5  45
```

```
arrange(nepali, desc(wt))[1:2, ]
```

```
##       id sex   wt    ht mage lit died alive age
## 1 360114   2 19.2 107.4   29   1    0     4  70
## 2 120561   2 18.9 105.7   35   0    4     8  59
```

slice

Grouping with mutate versus summarize

Quotation marks

Quotation marks

Related to this is the question of when you must use quotation marks. For example, if you are indexing using square brackets, you must use quotations when you reference column or row names:

```
worldcup[1:2, c("Shots", "Passes")]
```

```
##         Shots Passes
## Abdoun      0      6
## Abe         0    101
```

Quotation marks

If you do not, R looks for an object with that name. If it can't find it, it gives you an error:

```
worldcup[1:2, c(Shots, Passes)]
```

```
Error in `[.data.frame`(worldcup, 1:2, c(Shots, Passes)) :
  object 'Shots' not found
```

Quotation marks

If it can find it, it uses whatever's saved in that object to index:

```
Shots <- "Team"
Passes <- "Position"
worldcup[1:2, c(Shots, Passes)]
```

```
##              Team   Position
## Abdoun Algeria Midfielder
## Abe      Japan Midfielder
```

We will take advantage of this when we write loops and functions.

Quotation marks

There are, however, several examples of functions that ask you to name the dataframe, and then you don't have to include quotations around the column names.
For example:

```
mod_1 <- glm(Shots ~ Time, data = worldcup,
             family = poisson(link = "log"))
coef(mod_1)
```

```
##  (Intercept)         Time
## -0.094584866  0.003704373
```

Quotation marks

Other examples:

```
ggplot(worldcup, aes(x = Time, y = Shots)) + geom_point()
```

```
goalies <- subset(worldcup, Position == "Goalkeeper")
```

Note that, in all of these, you are specifying which dataframe to use. R will look in that dataframe first for a column with that name. If it can't find one, only then will it look for an object outside the dataframe with the name.
Many of the functions we'll use today fall under this category.