Programming day 4: Data tidying, merging, exporting

Princeton Sociology Methods Camp

Angela Li[1]

Princeton University

August 29, 2023

[1]These slides are the collective effort of everyone who has contributed to Methods Camp. Please see our website at https://pusocmethodscamp.org/#about for more information.

Feedback from last session

See Coding Feedback slides

Outline

1. Tidy data and reshaping from long to wide (and vice versa)
2. Saving and exporting data
3. Merging data: basic case and variations
4. Briefly: Useful packages and commands for integrating tables and figures in Rmarkdown or LaTeX

How to talk about data

1. A dataset is a collection of **values**. A value is the stuff in a cell. Each value belongs to a **variable** and an **observation**
2. A variable contains all values that measure the same underlying attribute across units
3. An observation contains all values measured on the same unit, across attributes.

Tidy Data

Three conditions for a tidy dataset[2]:

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

[2]Source: Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59(10)

Sad example: here is some information about how much sleep per night your instructors get, by year of grad school

Is this dataset tidy?

name	yeargrad	avgsleep
Angela	1	6
Angela	2	6
Angela	3	5
Varun	1	7
Varun	2	6
Varun	3	5

Sad example: here is some information about how much sleep per night your instructors get, by year of grad school

Is this dataset tidy?

name	Year1	Year2	Year3
Angela	6	6	5
Varun	7	6	5

Tidy datasets are all alike; every messy dataset is messy in its own way (Hadley Wickham quoting Leo Tolstoy)

Infinite number of ways that data can be messy, but here are five common problems:

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

This dataset exhibits which one of the common problems?

name	Year1	Year2	Year3
Angela	6	6	5
Varun	7	6	5

This dataset exhibits which one of the common problems?

name	Year1	Year2	Year3
Angela	6	6	5
Varun	7	6	5

Answer: Problem 1, Column headers are values not variables

Problem 2: Multiple variables in one column

Let's say University Health Services saw this data and wanted to investigate the variation in graduate student sleep patterns. They think that where students live and where their offices are might make a difference, so they've relabelled Angela as someone who works on Wallace's 1st floor and lives in Graduate Housing, and Varun as someone who works in Wallace's 2nd floor and lives off-campus.

year	W1_GH	W2_OC
1	6	7
2	6	6
3	5	5

Problem 3: Variables are stored in both rows and columns

The dean of graduate affairs caught wind of UHS' ongoing analyses and want to know why they are only investigating sleep patterns. The dean also wants to know about graduate students' exercise, drinking, and smoking behaviors. Due to rampant false reporting caused by social desirability bias, UHS was not able to collect reliable data for drinking and smoking, but they did get some data about avg hours of exercise per day. Unfortunately the data is formatted like this:

name	activity	Year1	Year2	Year3
Angela	sleep	6	6	5
Angela	exercise	1	0.5	0
Varun	sleep	7	6	5
Varun	exercise	2	0	0

Problem 4: Multiple types in one table

Sometimes you'll work with values that are collected at multiple levels. For example, while they are research *student*-level variation in sleep and exercise, the UHS might also be interested in getting access to existing data about teaching requirements for each *department*.

During tidying, each type of observational unit should be stored in its own table (e.g. tidy the individual-level table about sleep and exercise and tidy the department-level table about teaching requirements separately)

However, during analysis, working directly with relational data can be inconvenient, so we often merge datasets back into one table after tidying (we'll get to this later).

Problem 5: One type in multiple tables

This is kind of like the complement to Problem 4 – sometimes a single type of observational unit will have values spread over multiple tables. For example, suppose UHS surveyed students about only exercise because they already had data about sleep. Those two datasets are likely stored in different tables because they were collected at different times.

Tidying then depends on if the data structures in each table are consistent. If they are not, you should tidy each table (or format) separately. Once they are consistent, the "plyr" package is a good tool for compiling.

Tidying with tidyr: problem 1 (also known as "wide" to "long")

```
sleep_wide <- tibble(name = c("Angela", "Varun"),
                     year1 = c(6,7),
                     year2 = c(6,6),
                     year3 = c(5,5))
sleep_wide
```

```
## # A tibble: 2 x 4
##   name   year1 year2 year3
##   <chr>  <dbl> <dbl> <dbl>
## 1 Angela     6     6     5
## 2 Varun      7     6     5
```

Tidying with tidyr: wide to long

```
library(tidyverse)
sleep_long <- sleep_wide %>%
  pivot_longer(-name,
               names_to = "year",
               values_to = "avgsleep")
sleep_long
```

```
## # A tibble: 6 x 3
##   name   year  avgsleep
##   <chr>  <chr>    <dbl>
## 1 Angela year1        6
## 2 Angela year2        6
## 3 Angela year3        5
## 4 Varun  year1        7
## 5 Varun  year2        6
## 6 Varun  year3        5
```

tidyr::pivot_longer syntax deconstructed

```
pivot_longer(cols = c(year1, year2, year3),
             names_to = year,
             values_to = avgsleep)
```

- **cols**: (in this case, "year1", "year2", and "year3" OR "-id" - everything except for the "id" column) the columns that store the values you are pivoting
- **names_to**: the name of the new variable (whose values are the column headers)
- **values_to**: the name of the new variable for the underlying attribute that the values are measuring

tidyr::pivot_longer alternative syntax

Instead of writing out all the columns you want to pivot, you can also just specify which ones in the dataframe you DON'T want to pivot:

```
sleep_long <- sleep_wide %>%
  pivot_longer(-name,
               names_to = "year",
               values_to = "avgsleep")
sleep_long
```

```
## # A tibble: 6 x 3
##   name   year  avgsleep
##   <chr>  <chr>    <dbl>
## 1 Angela year1        6
## 2 Angela year2        6
## 3 Angela year3        5
## 4 Varun  year1        7
## 5 Varun  year2        6
## 6 Varun  year3        5
```

Switching back to "wide" with tidyr::pivot_wider

```
sleep_wide2 <- sleep_long %>%
  pivot_wider(names_from = year, values_from = avgsleep)

sleep_wide2
```

```
## # A tibble: 2 x 4
##   name   year1 year2 year3
##   <chr>  <dbl> <dbl> <dbl>
## 1 Angela     6     6     5
## 2 Varun      7     6     5
```

Tidying Problem 2: multiple variables in one column

Recall this problem:

```
## # A tibble: 3 x 3
##    year W1_GH W2_OC
##   <dbl> <dbl> <dbl>
## 1     1     6     7
## 2     2     6     6
## 3     3     5     5
```

Tidying multiple variables in one column

Multiple problems here: it's not just that the columns contain information about more than one variable, but the column headers are values, not variables (Problem 1 again), so we fix that first.

```
sleep_p2_tidy <- sleep_p2 %>%
  pivot_longer(cols = c(W1_GH, W2_OC),
               names_to = "office_housing",
               values_to = "avgsleep")

sleep_p2_tidy
```

```
## # A tibble: 6 x 3
##    year office_housing avgsleep
##   <dbl> <chr>              <dbl>
## 1     1 W1_GH                  6
## 2     1 W2_OC                  7
## 3     2 W1_GH                  6
## 4     2 W2_OC                  6
## 5     3 W1_GH                  5
## 6     3 W2_OC                  5
```

Using tidyr::separate to - you guessed it - separate one column into multiple

```
sleep_p2_tidy <- sleep_p2 %>%
  pivot_longer(cols = c(W1_GH, W2_OC),
               names_to = "office_housing",
               values_to = "avgsleep") %>%
  separate(col = office_housing, into = c("office", "housing"), sep = "_")

sleep_p2_tidy
```

```
## # A tibble: 6 x 4
##    year office housing avgsleep
##   <dbl> <chr>  <chr>      <dbl>
## 1     1 W1     GH             6
## 2     1 W2     OC             7
## 3     2 W1     GH             6
## 4     2 W2     OC             6
## 5     3 W1     GH             5
## 6     3 W2     OC             5
```

Tidyr::separate syntax deconstructed

```
separate(col = office_housing,
         into = c("office", "housing"),
         sep = "_")
```

- **col**: the name of the column you are trying to separate
- **into**: a character vector of the names of the new variables
- **sep**: character separator (in this case, "_") interpreted as regular expression if character and position if numeric. Other common character separators include "." and ""
- **remove**: default is TRUE so we didn't type it out here. If you want to keep the input column even after separating, set remove to FALSE

The opposite of separate is unite

Let's say we actually have information about one variable split across columns. For example, you get data about office location but building is recorded in one column and floor on another, and you only care about the combination of the two.

```
library(stringr)
sleep_pls_unite <- sleep_p2_tidy %>%
  mutate(building = stringr::str_sub(office, 1, 1),
         floor = stringr::str_sub(office, -1, -1)) %>%
  select(year, building, floor, housing, avgsleep)

sleep_pls_unite
```

```
## # A tibble: 6 x 5
##    year building floor housing avgsleep
##   <dbl> <chr>    <chr> <chr>      <dbl>
## 1     1 W        1     GH             6
## 2     1 W        2     OC             7
## 3     2 W        1     GH             6
## 4     2 W        2     OC             6
## 5     3 W        1     GH             5
## 6     3 W        2     OC             5
```

```
sleep_united <- sleep_pls_unite %>%
  unite(col = "office", building, floor, sep = "")

sleep_united
```

Tidying Problem 3: Variables stored in both rows and columns

```
sleep_p3 <- tibble(name = c(rep("Angela",2), rep("Varun", 2)),
                   activity = rep(c("sleep", "exercise"), 2),
                   year1 = c(6,1,7,2),
                   year2 = c(6, 0.5, 6, 0),
                   year3 = c(5,0,5,0))
sleep_p3
```

```
## # A tibble: 4 x 5
##   name   activity year1 year2 year3
##   <chr>  <chr>    <dbl> <dbl> <dbl>
## 1 Angela sleep        6   6       5
## 2 Angela exercise     1   0.5     0
## 3 Varun  sleep        7   6       5
## 4 Varun  exercise     2   0       0
```

Tidying variables stored in both rows and columns

Identify the problems: 1. Columns `year1`, `year2`, `year3` are values, should be pivoted into one variable 2. Values of the column `activity` actually represent variables, need to spread into two columns

Step 1: Pivot year columns into one variable

```
sleep_p3_tidy <- sleep_p3 %>%
  pivot_longer(cols = c(year1, year2, year3), names_to = "year", values_to = "avgtime")

sleep_p3_tidy
```

```
## # A tibble: 12 x 4
##    name   activity year  avgtime
##    <chr>  <chr>    <chr>   <dbl>
##  1 Angela sleep    year1     6
##  2 Angela sleep    year2     6
##  3 Angela sleep    year3     5
##  4 Angela exercise year1     1
##  5 Angela exercise year2     0.5
##  6 Angela exercise year3     0
##  7 Varun  sleep    year1     7
##  8 Varun  sleep    year2     6
##  9 Varun  sleep    year3     5
## 10 Varun  exercise year1     2
## 11 Varun  exercise year2     0
## 12 Varun  exercise year3     0
```

Tidying variables stored in both rows and columns

Step 2: Spread sleep and exercise into columns, with avgtime as values (pipe it!)

```
sleep_p3_tidy <- sleep_p3 %>%
  pivot_longer(cols = c(year1, year2, year3),
               names_to = "year",
               values_to = "avgtime") %>%
  pivot_wider(names_from = "activity",
              values_from = "avgtime")

sleep_p3_tidy
```

```
## # A tibble: 6 x 4
##   name   year  sleep exercise
##   <chr>  <chr> <dbl>    <dbl>
## 1 Angela year1     6      1
## 2 Angela year2     6      0.5
## 3 Angela year3     5      0
## 4 Varun  year1     7      2
## 5 Varun  year2     6      0
## 6 Varun  year3     5      0
```

Exporting Data

- You just wrote a bunch of tidying code that you don't want to run every time you do analysis.

Exporting Data

- You just wrote a bunch of tidying code that you don't want to run every time you do analysis.
- And often time, you will need to transform your data in multiple ways to work on multiple types of analyses.

Exporting Data

- You just wrote a bunch of tidying code that you don't want to run every time you do analysis.
- And often time, you will need to transform your data in multiple ways to work on multiple types of analyses.
- It is convenient and conducive to reproducibility to "save" your new tidy dataset: exporting it by writing it to a new file that you can load directly the next time you need it.

Exporting Data

- You just wrote a bunch of tidying code that you don't want to run every time you do analysis.
- And often time, you will need to transform your data in multiple ways to work on multiple types of analyses.
- It is convenient and conducive to reproducibility to "save" your new tidy dataset: exporting it by writing it to a new file that you can load directly the next time you need it.
- (But you should still *always* save the code you wrote to transform the raw/original data into its new form.)

Exporting Data

- Export command depends on the type of file you are trying to write to
  - write.csv for CSV, write.xslx for Excel spreadsheet, write.dta for Stata file, etc.
- When exporting, do NOT use the same name as the original data – you'll write over it
- Be default, the new file will be saved in your current working directory. If you want to save it elsewhere, specify the path name

```
#Example: saving csv file to "output" folder in my current working directory
write_csv(sleep_p3_tidy, "output/sleep_p3_tidy.csv")

#Example, saving Stata file to my Downloads folder
library(haven)
write_dta(sleep_long, "~/Downloads/gss_long.dta")
```

Basic merge

We hypothesize that the number of courses a grad student takes affects how much they sleep, so we get a bit more information on our instructors' semester course schedule from the University Registrar.

However, they only release the first two years of coursework, and they accidentally included another person's course information in the released data (oops! data privacy issue!)

```
coursework <- tibble(name = c(rep("Angela",4), rep("Varun", 4), rep("Matt",4)),
                     year = rep(c("year1", "year1", "year2", "year2"), 3),
                     semester = rep(c(1,2), 6),
                     classes = c(4,4.5,3,3,4,4,4,3,4,3.5,4,3))
```

Basic merge

```
coursework
```

```
## # A tibble: 12 x 4
##    name   year  semester classes
##    <chr>  <chr>    <dbl>   <dbl>
##  1 Angela year1        1     4
##  2 Angela year1        2     4.5
##  3 Angela year2        1     3
##  4 Angela year2        2     3
##  5 Varun  year1        1     4
##  6 Varun  year1        2     4
##  7 Varun  year2        1     4
##  8 Varun  year2        2     3
##  9 Matt   year1        1     4
## 10 Matt   year1        2     3.5
## 11 Matt   year2        1     4
## 12 Matt   year2        2     3
```

Basic merge

We can merge this with our tidy data as follows. The typical merge you might want to do in R is a `left_join`, keep all rows from "left" table even if observation doesn't have matching row in "right" table. Note that we dropped the extra person from the joining data.

```
leftjoin <- left_join(sleep_p3_tidy, coursework, by = c("name", "year"))
leftjoin
```

```
## # A tibble: 10 x 6
##    name   year  sleep exercise semester classes
##    <chr>  <chr> <dbl>    <dbl>    <dbl>   <dbl>
##  1 Angela year1     6      1          1     4
##  2 Angela year1     6      1          2     4.5
##  3 Angela year2     6      0.5        1     3
##  4 Angela year2     6      0.5        2     3
##  5 Angela year3     5      0         NA    NA
##  6 Varun  year1     7      2          1     4
##  7 Varun  year1     7      2          2     4
##  8 Varun  year2     6      0          1     4
##  9 Varun  year2     6      0          2     3
## 10 Varun  year3     5      0         NA    NA
```

Complication of basic merge: observations missing!

- A good habit after merging is to compare the number of rows in the original data with the number of rows in the new merged dataset–if the number of rows either increases or decreases, you'll want to investigate
- In this case, doing this reveals that during our merge, doubled our observation count!
- How do we: 1) find out what happened, 2) correct if necessary?

How should we think about joining this data?

1. Decide that anything that doesn't have a match should dropped during the merge - we only want to do our analysis on years 1 and 2. Only keep rows of the first data.frame that have corresponding records in the second data.frame - sometimes called *inner join*:

$$sleepdata \cap coursedata$$

2. Decide to keep those observations even if their values are not in the second data.frame. There are a variety of combinations for this option, which we'll review next (as a set, these are sometimes known as *outer joins*)[3]

$$sleepdata \cup coursedata$$

[3] The language of inner join and outer join come from SQL, which is a domain-specific language used for managing relational database systems.

Different join options: inner join

Only keep observations in "sleep_p3_tidy" that have matching observations in "coursework"

```
onlycommon <- inner_join(sleep_p3_tidy, coursework, by = c("name", "year"))

onlycommon
```

```
## # A tibble: 8 x 6
##   name   year  sleep exercise semester classes
##   <chr>  <chr> <dbl>    <dbl>    <dbl>   <dbl>
## 1 Angela year1     6      1          1     4
## 2 Angela year1     6      1          2     4.5
## 3 Angela year2     6      0.5        1     3
## 4 Angela year2     6      0.5        2     3
## 5 Varun  year1     7      2          1     4
## 6 Varun  year1     7      2          2     4
## 7 Varun  year2     6      0          1     4
## 8 Varun  year2     6      0          2     3
```

Different join options: full join

Keep all observations from each data frame. Note that we kept "year3" from the sleep data even though we don't have any observations from that year in "coursework". We also kept all of the observations that don't have matching sleep data.

```
keepallobs <- full_join(sleep_p3_tidy, coursework, by = c("name", "year"))

keepallobs
```

```
## # A tibble: 14 x 6
##    name   year  sleep exercise semester classes
##    <chr>  <chr> <dbl>    <dbl>    <dbl>   <dbl>
##  1 Angela year1     6      1          1     4
##  2 Angela year1     6      1          2     4.5
##  3 Angela year2     6      0.5        1     3
##  4 Angela year2     6      0.5        2     3
##  5 Angela year3     5      0         NA    NA
##  6 Varun  year1     7      2          1     4
##  7 Varun  year1     7      2          2     4
##  8 Varun  year2     6      0          1     4
##  9 Varun  year2     6      0          2     3
## 10 Varun  year3     5      0         NA    NA
## 11 Matt   year1    NA     NA          1     4
## 12 Matt   year1    NA     NA          2     3.5
## 13 Matt   year2    NA     NA          1     4
## 14 Matt   year2    NA     NA          2     3
```

Different join options: right join

Keep all rows from "right" table even if observation doesn't have matching row in "left" table. Note that now we retain observations for the third person but dropped year 3 for Angela and Varun.

```
keeprightrows <- right_join(sleep_p3_tidy, coursework, by = c("name", "year"))

keeprightrows
```

```
## # A tibble: 12 x 6
##    name   year  sleep exercise semester classes
##    <chr>  <chr> <dbl>    <dbl>    <dbl>   <dbl>
##  1 Angela year1     6      1          1     4
##  2 Angela year1     6      1          2     4.5
##  3 Angela year2     6      0.5        1     3
##  4 Angela year2     6      0.5        2     3
##  5 Varun  year1     7      2          1     4
##  6 Varun  year1     7      2          2     4
##  7 Varun  year2     6      0          1     4
##  8 Varun  year2     6      0          2     3
##  9 Matt   year1    NA     NA          1     4
## 10 Matt   year1    NA     NA          2     3.5
## 11 Matt   year2    NA     NA          1     4
## 12 Matt   year2    NA     NA          2     3
```

Integrating tables in RMarkdown and LaTeX

- We've been printing various data.frames, tables, and tibbles in our R code chunks, but these objects are not the best looking.

Integrating tables in RMarkdown and LaTeX

- We've been printing various data.frames, tables, and tibbles in our R code chunks, but these objects are not the best looking.
- Or, what if we want to recreate some results from analyses in the LaTeX environment without having to copy/paste all the numbers, which creates a lot of room for errors?

Integrating tables in RMarkdown and LaTeX

- We've been printing various data.frames, tables, and tibbles in our R code chunks, but these objects are not the best looking.
- Or, what if we want to recreate some results from analyses in the LaTeX environment without having to copy/paste all the numbers, which creates a lot of room for errors?
- A couple popular packages (many out there): stargazer, xtable, kable. Most of them operate on *pandoc* magic – a free software that can convert files from Markdown (and other) formats into HTML, TeX, and PDF via LaTeX (and other) formats.

Integrating tables in RMarkdown and LaTex: two common ways

- **Option 1**: run packages like stargazer, xtable, and kable in R file and get LaTeX code output, which you can then copy/paste into a TeX editor (including collaborative online hosts like Overleaf). You can also manually modify the LaTeX code this way.

Integrating tables in RMarkdown and LaTex: two common ways

- **Option 1**: run packages like stargazer, xtable, and kable in R file and get LaTeX code output, which you can then copy/paste into a TeX editor (including collaborative online hosts like Overleaf). You can also manually modify the LaTeX code this way.
- **Option 2**: use these packages in the R code chunks of a Rmd file like the ones you've been writing, and add the option **results = 'asis'** at the beginning of the chunk. Then, when you knit, your table objects will be converted to PDF via LaTeX format. You can also add the option **echo = FALSE** at the beginning of the chunk if you want to display just the table and not the underlying code that produced it (though please show all of your code in homework assignments!)

Example: stargazer package

```
library(stargazer)
#print summary table of age for coursework data
stargazer((leftjoin %>% select(exercise, sleep)), header = FALSE, title = " Summary table",
          font.size = "tiny")
```

Table 1: Summary table

Statistic	N	Mean	St. Dev.	Min	Max

Example: stargazer package

```
#not a sensical regression in this example but used to illustrate
reg <- glm(sleep ~ exercise + classes, data  = leftjoin)
stargazer(reg, header = FALSE, title = " Regression results",
          font.size = "tiny")
```

Table 2: Regression results

	Dependent variable:
	sleep
exercise	0.566^{***}
	(0.139)
classes	-0.134
	(0.185)
Constant	6.248^{***}
	(0.634)
Observations	8
Log Likelihood	0.646
Akaike Inf. Crit.	4.709
Note:	*p$<$0.1; **p$<$0.05; ***p$<$0.01

Example: xtable package

```
library(xtable)
xtable(sleep_long)
```

% latex table generated in R 4.2.1 by xtable 1.8-4 package % Mon Sep 4 15:54:08 2023

	name	year	avgsleep
1	Angela	year1	6.00
2	Angela	year2	6.00
3	Angela	year3	5.00
4	Varun	year1	7.00
5	Varun	year2	6.00
6	Varun	year3	5.00

Example: kable (knitr package)

```
library(knitr)
kable(sleep_long)
```

name	year	avgsleep
Angela	year1	6
Angela	year2	6
Angela	year3	5
Varun	year1	7
Varun	year2	6
Varun	year3	5