Day 3: Data Manipulation

Sociology Methods Camp

September 6th, 2018

Outline

1. Tidy data and reshaping from long to wide (and vice versa)

Outline

1. Tidy data and reshaping from long to wide (and vice versa)
2. Saving and exporting data

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

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

For a practice example later, we'll use data from the General Social Survey (GSS) to investigate homophily in social networks

Figure 1

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**

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

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[1]:

1. Each variable forms a column

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

Tidy Data

Three conditions for a tidy dataset[1]:

1. Each variable forms a column
2. Each observation forms a row

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

Tidy Data

Three conditions for a tidy dataset[1]:

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

[1]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
Katie	1	6
Katie	2	6
Katie	3	5
Xinyi	1	7
Xinyi	2	6
Xinyi	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
Katie	6	6	5
Xinyi	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

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

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

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

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
Katie	6	6	5
Xinyi	7	6	5

This dataset exhibits which one of the common problems?

name	Year1	Year2	Year3
Katie	6	6	5
Xinyi	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 Katie as someone who works on Wallace's 1st floor and lives in Graduate Housing, and Xinyi as someone who works in Wallace's 1st floor and lives off-campus.

year	W2_GH	W1_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
Katie	sleep	6	6	5
Katie	exercise	1	0.5	0
Xinyi	sleep	7	6	5
Xinyi	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”)

```r
sleep.wide <- data.frame(name = c("Katie", "Xinyi"),
                         year1 = c(6,7),
                         year2 = c(6,6),
                         year3 = c(5,5))
sleep.wide
```

```
##    name year1 year2 year3
## 1 Katie     6     6     5
## 2 Xinyi     7     6     5
```

Tidying with tidyr: wide to long

```r
library(tidyverse)
library(tidyr)
library(magrittr)
library(dplyr)
sleep.long <- sleep.wide %>%
  gather(key = year, value = avgsleep, -name)
sleep.long
```

```
##    name  year avgsleep
## 1 Katie year1        6
## 2 Xinyi year1        7
## 3 Katie year2        6
## 4 Xinyi year2        6
## 5 Katie year3        5
## 6 Xinyi year3        5
```

tidyr::gather syntax deconstructed

```
gather(key = year, value = avgsleep, year1, year2, year3)
```

- **key**: the name of the new variable (whose values are the column headers)
- **value**: the name of the underlying attribute that the values are measuring
- **other arguments**: (in this case, "year1", "year2", and "year3") the columns that store the values you are gathering

tidyr::gather alternative syntax

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

```
sleep.long <- sleep.wide %>%
  gather(year, avgsleep, -name)
sleep.long
```

```
##    name  year avgsleep
## 1 Katie year1        6
## 2 Xinyi year1        7
## 3 Katie year2        6
## 4 Xinyi year2        6
## 5 Katie year3        5
## 6 Xinyi year3        5
```

Switching back to “wide” with tidyr::spread

```
sleep.wide2 <- sleep.long %>%
  spread(key = year, value = avgsleep)
sleep.wide2
```

```
##    name year1 year2 year3
## 1 Katie     6     6     5
## 2 Xinyi     7     6     5
```

Alternatives to tidyr package

1. "reshape" package (base R)

Alternatives to tidyr package

1. "reshape" package (base R)

1. originally designed for longitudinal data/repeated measurements

Alternatives to tidyr package

1. "reshape" package (base R)

1. originally designed for longitudinal data/repeated measurements
2. use the "direction" argument to indicate whether you are going "long" or "wide"

Alternatives to tidyr package

1. "reshape" package (base R)

1. originally designed for longitudinal data/repeated measurements
2. use the "direction" argument to indicate whether you are going "long" or "wide"

2. "reshape2" package

Alternatives to tidyr package

1. "reshape" package (base R)

1. originally designed for longitudinal data/repeated measurements
2. use the "direction" argument to indicate whether you are going "long" or "wide"

2. "reshape2" package

1. Hadley Wickham's reboot of "reshape" but not part of tidyverse

Alternatives to tidyr package

1. "reshape" package (base R)

1. originally designed for longitudinal data/repeated measurements
2. use the "direction" argument to indicate whether you are going "long" or "wide"

2. "reshape2" package

1. Hadley Wickham's reboot of "reshape" but not part of tidyverse
2. tidyr:gather::reshape2:melt , tidyr:spread::reshape2:cast [2]

[2] more detailed comparison here http://www.milanor.net/blog/reshape-data-r-tidyr-vs-reshape2/

Tidying Problem 2: multiple variables in one column

Recall this problem:

```
##   year W2_GH W1_OC
## 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 %>%
  gather(key = OfficeHousing, value = avgsleep, W2_GH, W1_OC)
sleep.p2.tidy
```

```
##   year OfficeHousing avgsleep
## 1    1         W2_GH        6
## 2    2         W2_GH        6
## 3    3         W2_GH        5
## 4    1         W1_OC        7
## 5    2         W1_OC        6
## 6    3         W1_OC        5
```

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

```
sleep.p2.tidy <- sleep.p2 %>%
  gather(key = OfficeHousing, value = avgsleep, W2_GH, W1_OC) %>%
  separate(col = OfficeHousing, into = c("Office", "Housing"), sep = "_")
sleep.p2.tidy
```

```
##   year Office Housing avgsleep
## 1    1     W2      GH        6
## 2    2     W2      GH        6
## 3    3     W2      GH        5
## 4    1     W1      OC        7
## 5    2     W1      OC        6
## 6    3     W1      OC        5
```

Tidyr::separate syntax deconstructed

```
separate(col = OfficeHousing, 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**: (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
```

```
##   year Building Floor Housing avgsleep
## 1    1        W     2      GH        6
## 2    2        W     2      GH        6
## 3    3        W     2      GH        5
## 4    1        W     1      OC        7
## 5    2        W     1      OC        6
## 6    3        W     1      OC        5
```

```
sleep.united <- sleep.pls.unite %>%
  unite(col = "Office", Building, Floor, sep = "")
sleep.united
```

```
##   year Office Housing avgsleep
## 1    1     W2      GH        6
## 2    2     W2      GH        6
## 3    3     W2      GH        5
```

Tidying Problem 3: Variables stored in both rows and columns

```
sleep.p3 <- data.frame(name = c(rep("Katie",2), rep("Xinyi", 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
```

```
##    name activity Year1 Year2 Year3
## 1 Katie    sleep     6   6.0     5
## 2 Katie exercise     1   0.5     0
## 3 Xinyi    sleep     7   6.0     5
## 4 Xinyi exercise     2   0.0     0
```

Tidying variables stored in both rows and columns

Identify the problems: 1. Columns Year1, Year2, Year3 are values, should be gathered into one variable 2. Values of the column "activity" actually represent variables, need to spread into two columns

Step 1: Gather year columns into one variable

```
sleep.p3.tidy <- sleep.p3 %>%
  gather(key = year, value = avgtime, Year1, Year2, Year3)
sleep.p3.tidy
```

```
##     name activity  year avgtime
## 1  Katie    sleep Year1     6.0
## 2  Katie exercise Year1     1.0
## 3  Xinyi    sleep Year1     7.0
## 4  Xinyi exercise Year1     2.0
## 5  Katie    sleep Year2     6.0
## 6  Katie exercise Year2     0.5
## 7  Xinyi    sleep Year2     6.0
## 8  Xinyi exercise Year2     0.0
## 9  Katie    sleep Year3     5.0
## 10 Katie exercise Year3     0.0
## 11 Xinyi    sleep Year3     5.0
## 12 Xinyi exercise Year3     0.0
```

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 %>%
  gather(key = year, value = avgtime, Year1, Year2, Year3) %>%
  spread(key = "activity", value = "avgtime")
sleep.p3.tidy
```

```
##    name  year exercise sleep
## 1 Katie Year1      1.0     6
## 2 Katie Year2      0.5     6
## 3 Katie Year3      0.0     5
## 4 Xinyi Year1      2.0     7
## 5 Xinyi Year2      0.0     6
## 6 Xinyi Year3      0.0     5
```

Now let's practice on a more realistic example

Every once in a while, the GSS ask survey respondents to list up to five people they discuss important matters with[3], as well as demographic information about these confidants.

To see how this data is stored, load in the csv file called "gss.reshape.example.csv" and view the first two observations of the dataset

[3] There is a great deal of debate about the reliability of this question wording for measuring social networks

Have social networks grown more homophiliac over time?

Smith, Jeffrey A., Miller McPherson, and Lynn Smith-Lovin. 2014. "Social Distance in the United States: Sex, Race, Religion, Age, and Education Homophily among Confidants, 1985-2004." *American Sociological Review* 79(3):432-456

"We use data from the 1985 and 2004 General Social Surveys to ask whether the strengths of five social distinctions—sex, race/ethnicity, religious affiliation, age, and education—changed over the past two decades in core discussion networks."

Let's examine a simplified version of this question: find the average age difference among core discussion social ties in 1985 and the average age difference among core discussion social ties in 2004. (Note: to actually make inference about if the difference between these averages are meaningful, we'd have to control for demographic differences in the U.S. population in those two years, which we won't do here.)

To make turn the unit of analysis from respondent to ties, we have to reorganize the data

Your turn to try!

Now calculate average age difference

1. Filter by year

Now calculate average age difference

1. Filter by year
2. For each year, calculate difference between ego's age and alter's age for all ties

Now calculate average age difference

1. Filter by year
2. For each year, calculate difference between ego's age and alter's age for all ties
3. Take absolute value of each tie's age difference (why?)

Now calculate average age difference

1. Filter by year
2. For each year, calculate difference between ego's age and alter's age for all ties
3. Take absolute value of each tie's age difference (why?)
4. Find average (and standard deviation, if you're the overachieving sort)

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

```
#Example: saving csv file to my current working directory
write.csv(long, "gss_long.csv")

#Example, saving Stata file to my Downloads folder
write.dta(long, "~/Downloads/gss_long.dta")
```

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.

```
#Example: saving csv file to my current working directory
write.csv(long, "gss_long.csv")

#Example, saving Stata file to my Downloads folder
write.dta(long, "~/Downloads/gss_long.dta")
```

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

```
#Example: saving csv file to my current working directory
write.csv(long, "gss_long.csv")

#Example, saving Stata file to my Downloads folder
write.dta(long, "~/Downloads/gss_long.dta")
```

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 my current working directory
write.csv(long, "gss_long.csv")

#Example, saving Stata file to my Downloads folder
write.dta(long, "~/Downloads/gss_long.dta")
```

Merging Data

- Earlier we noted that you can't make much meaningful inference by comparing average age differences between two years because the probability of having a small or large average age difference depends on the age distribution of the population in those two years. How would you manage to take population distributions into account?

Merging Data

- Earlier we noted that you can't make much meaningful inference by comparing average age differences between two years because the probability of having a small or large average age difference depends on the age distribution of the population in those two years. How would you manage to take population distributions into account?
- For example, you wonder, if Judaism is a rarer religion that Protestant Christianity, does that make it more or less likely that two Jewish people will form a tie than two Protestant people?

Merging Data

- Earlier we noted that you can't make much meaningful inference by comparing average age differences between two years because the probability of having a small or large average age difference depends on the age distribution of the population in those two years. How would you manage to take population distributions into account?
- For example, you wonder, if Judaism is a rarer religion that Protestant Christianity, does that make it more or less likely that two Jewish people will form a tie than two Protestant people?
- Because the GSS is nationally representative, we can get the size of each religious group. In "wide" format, the unit of observation is an individual, so we can append the size his/her religious group as a variable.

Generating the data on religious group size

```r
#find religious proportions
relig.prop <- prop.table(table(wide$RELIG))
relig.prop
```

```
##
##   Catholic     Jewish       None      Other Protestant
## 0.24858002 0.02338791 0.10457735 0.04744404 0.57601069
```

```r
#check class
class(relig.prop)
```

```
## [1] "table"
```

```r
#convert to data.frame
relig.prop.df <- as.data.frame(relig.prop)
relig.prop.df
```

```
##         Var1       Freq
## 1   Catholic 0.24858002
## 2     Jewish 0.02338791
## 3       None 0.10457735
## 4      Other 0.04744404
## 5 Protestant 0.57601069
```

```r
colnames(relig.prop.df) <- c("RELIG", "religprop")
relig.prop.df
```

Basic merge

The following code merges using the RELIG column present in both the x data.frame (wide) and the y data.frame (relig.prop.df)

```
#add clearer ids to df
wide$idnew <- 1:nrow(wide)

#filter to first 2 obs to see before merge
wide  %>%
  filter(idnew == 1 | idnew == 2)
```

```
##   year    id_ AGE       EDUC  SEX RACETH    RELIG   sex1 race1   relig1
## 1 1985 1985.1  33 16 or more Male  White   Jewish   Male White   Jewish
## 2 1985 1985.2  49 16 or more Male  White Catholic Female White Catholic
##       educ1 age1   sex2 race2     relig2     educ2 age2 sex3 race3
## 1 16 or more   32 Female White Protestant 16 or more   29 Male White
## 2   12 years   42   Male White     Jewish 16 or more   44 Male White
##   relig3      educ3 age3   sex4 race4   relig4      educ4 age4   sex5
## 1 Jewish 16 or more   32   Male White   Jewish 16 or more   35 Female
## 2 Jewish 16 or more   45 Female White Catholic   12 years   40   Male
##   race5   relig5       educ5 age5 idnew
## 1 White Catholic 13-15 years   29     1
## 2 White   Jewish  16 or more   50     2
```

Basic merge

```
#merge wide and relig.prop.df by their common column name RELIG.
widewithprop <- merge(wide, relig.prop.df, by = "RELIG")

#view same two observations
widewithprop %>%
  filter((id_ == 1985.1 & AGE ==  33) |
        (id_ == 1985.2 & AGE == 49)) %>%
  select(RELIG, id_, AGE, EDUC, SEX, RACETH, idnew, religprop)
```

```
##      RELIG    id_ AGE        EDUC  SEX RACETH idnew  religprop
## 1 Catholic 1985.2  49 16 or more Male  White     2 0.24858002
## 2   Jewish 1985.1  33 16 or more Male  White     1 0.02338791
```

Complication of basic merge: different names for key/id to merge by in two data frames

What if the relig.prop.df had called the variable indicating the religious category, RELIGCAT instead of RELIG?

```
#rename column
relig.prop.df
```

```
##        RELIG  religprop
## 1   Catholic 0.24858002
## 2     Jewish 0.02338791
## 3       None 0.10457735
## 4      Other 0.04744404
## 5 Protestant 0.57601069
```

```
colnames(relig.prop.df)[1] <- "RELIGCAT"
relig.prop.df
```

```
##     RELIGCAT  religprop
## 1   Catholic 0.24858002
## 2     Jewish 0.02338791
## 3       None 0.10457735
## 4      Other 0.04744404
## 5 Protestant 0.57601069
```

Different names for key/id to merge by in two data frames: Solution

```
#do the same merge
widewithprop2 <- merge(wide, relig.prop.df,
                       by.x = "RELIG", by.y = "RELIGCAT")

#view same two observations
widewithprop2 %>%
  filter(idnew == 1 | idnew == 2) %>%
  select(RELIG, id_, AGE, EDUC, SEX, RACETH, idnew, religprop)
```

```
##      RELIG    id_ AGE       EDUC  SEX RACETH idnew  religprop
## 1 Catholic 1985.2  49 16 or more Male  White     2 0.24858002
## 2   Jewish 1985.1  33 16 or more Male  White     1 0.02338791
```

Complication of basic merge: observations disappearing!

- 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, we lost 13 observations
- How do we: 1) find out who we lost, 2) correct if necessary?

Observations disappearing: Solution

```
#original count of obs and #obs after first merge
nrow(wide); nrow(widewithprop)
```

```
## [1] 3006
```

```
## [1] 2993
```

```
#one way to find out who we lost is to subset original data to just show the people
#whose ids appear in the pre-merge data but not in the post-merge data
lostinmerge <- wide %>%
  filter(!idnew %in% widewithprop2$idnew)

  wide[!wide$idnew %in% widewithprop2$idnew, ]
```

```
##      year      id_ AGE         EDUC    SEX   RACETH RELIG   sex1    race1
## 333  1985 1985.333  30     12 years Female    White  <NA> Female    White
## 346  1985 1985.346  40   16 or more   Male    White  <NA> Female    White
## 367  1985 1985.367  33     12 years   Male    Black  <NA>   Male    Black
## 735  1985 1985.735  60 Less than 10 Female    Black  <NA>   <NA>     <NA>
## 741  1985 1985.741  56  13-15 years Female    Black  <NA>   <NA>     <NA>
## 1554 2004 2004.350  47   16 or more Female    White  <NA>   Male    White
## 1653 2004 2004.223  55   16 or more   Male Hispanic  <NA> Female Hispanic
## 1670 2004 2004.256  40   16 or more Female    White  <NA>   <NA>     <NA>
## 1816 2004 2004.504  43     12 years   Male    White  <NA>   <NA>     <NA>
## 2097 2004 2004.107  49  13-15 years Female    White  <NA> Female    White
## 2476 2004 2004.181  55     12 years Female    White  <NA> Female    White
## 2850 2004 2004.256  63  13-15 years Female    White  <NA>   <NA>     <NA>
```

How should we treat these observations?

1. Decide that since their religion value (NA) is not in the second data.frame, that they should be dropped during the merge. More formally, the default of merge is to only keep rows of the first data.frame that have corresponding records in the second data.frame (so in this case, only GSS observations whose religious group is in the second data.frame). Sometimes called *inner join*:

$$gssdata \cap religiondata$$

$$gssdata \cup religiondata$$

How should we treat these observations?

1. Decide that since their religion value (NA) is not in the second data.frame, that they should be dropped during the merge. More formally, the default of merge is to only keep rows of the first data.frame that have corresponding records in the second data.frame (so in this case, only GSS observations whose religious group is in the second data.frame). Sometimes called *inner join*:

$$gssdata \cap religiondata$$

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*)[4]

$$gssdata \cup religiondata$$

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

Illustrating each option with more manageable data

```
simpledf <- data.frame(id = 1:4,
                       RELIG = c("Jewish", "Protestant",
                                 "Satanism", "Catholic"))
simpledf
```

```
##   id      RELIG
## 1  1     Jewish
## 2  2 Protestant
## 3  3   Satanism
## 4  4   Catholic
```

```
relig.prop.df
```

```
##     RELIGCAT  religprop
## 1   Catholic 0.24858002
## 2     Jewish 0.02338791
## 3       None 0.10457735
## 4      Other 0.04744404
## 5 Protestant 0.57601069
```

Different join options: inner join

Only keep observations in “simpledf” that have matching observations in “relig.prop.df”

```
onlycommon <- merge(simpledf, relig.prop.df,
                    by.x = "RELIG",
                    by.y = "RELIGCAT")
onlycommon
```

```
##        RELIG id  religprop
## 1   Catholic  4 0.24858002
## 2     Jewish  1 0.02338791
## 3 Protestant  2 0.57601069
```

Different join options: full outer join

Keep all observations from each data frame. Note that we kept “Satanism” from “simpldf” despite not having a proportion for that religion in “relig.prop.df”, and we retain the proportions for “none/other” even though we don’t have any observations in those categories in “simpledf”.

```
keepallobs <- merge(simpledf, relig.prop.df,
                    by.x = "RELIG",
                    by.y = "RELIGCAT",
                    all = TRUE)
keepallobs
```

```
##        RELIG id  religprop
## 1   Catholic  4 0.24858002
## 2     Jewish  1 0.02338791
## 3 Protestant  2 0.57601069
## 4   Satanism  3         NA
## 5       None NA 0.10457735
## 6      Other NA 0.04744404
```

Different join options: left outer join

Keep all rows from "left" table (simpledf in this case), even if observation does not have matching row in "right" (relig.prop.df). Note that we kept "Satanism" but dropped the proportions for "none/other".

```
keepleftrows <- merge(simpledf, relig.prop.df,
                      by.x = "RELIG",
                      by.y = "RELIGCAT",
                      all.x  = TRUE)
keepleftrows
```

```
##        RELIG id  religprop
## 1   Catholic  4 0.24858002
## 2     Jewish  1 0.02338791
## 3 Protestant  2 0.57601069
## 4   Satanism  3         NA
```

Different join options: right outer join

Keep all rows from "right" table (relig.prop.df) even if observation doesn't have matching row in "left" table (simpledf). Note that now we retain proportions for "none/other" but dropped Satanism.

```
keeprightrows <- merge(simpledf, relig.prop.df,
                       by.x = "RELIG",
                        by.y = "RELIGCAT",
                        all.y  = TRUE)
keeprightrows
```

```
##         RELIG id  religprop
## 1    Catholic  4 0.24858002
## 2      Jewish  1 0.02338791
## 3 Protestant  2 0.57601069
## 4        None NA 0.10457735
## 5       Other NA 0.04744404
```

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 wide gss data
stargazer((wide %>% select(AGE, year)), header = FALSE, title = " Summary table",
          font.size = "tiny")
```

Table 1: Summary table

Statistic	N	Mean	St. Dev.	Min	Max
AGE	2,994	45.836	17.262	18	89
year	3,006	1,994.304	9.500	1,985	2,004

Example: stargazer package

```r
#not a sensical regression in this example but used to illustrate
reg <- glm(RELIG ~ AGE + SEX, data  = wide, family = binomial(link = logit))
stargazer(reg, header = FALSE, title = " Regression results",
          font.size = "tiny")
```

Table 2: Regression results

	Dependent variable:
	RELIG
AGE	0.0002
	(0.002)
SEXMale	−0.056
	(0.085)
Constant	1.119***
	(0.126)
Observations	2,981
Log Likelihood	−1,672.521
Akaike Inf. Crit.	3,351.043
Note:	*p<0.1; **p<0.05; ***p<0.01

Example: xtable package

```
library(xtable)
xtable(sleep.long)
```

% latex table generated in R 3.3.1 by xtable 1.8-2 package % Thu Sep 6 11:43:28 2018

	name	year	avgsleep
1	Katie	year1	6.00
2	Xinyi	year1	7.00
3	Katie	year2	6.00
4	Xinyi	year2	6.00
5	Katie	year3	5.00
6	Xinyi	year3	5.00

Example: kable (knitr package)

```
library(knitr)
kable(sleep.long)
```

name	year	avgsleep
Katie	year1	6
Xinyi	year1	7
Katie	year2	6
Xinyi	year2	6
Katie	year3	5
Xinyi	year3	5

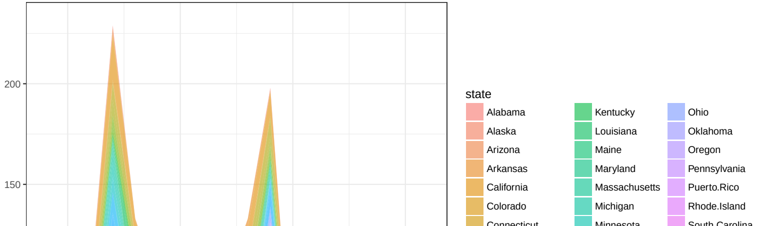
ggplot 2

- Datacamp has 2 great module on plotting

```
setwd("~/Dropbox/MethodsCamp/2018/Programming Lectures/Day2Programming/Assignment")
library(tidyverse)
dirty <- read.csv("stateyearreports.csv")
stateyear <- dirty %>% gather(state, reports, -c(1:2))

state.plot <- stateyear %>%
  ggplot(aes(x=year, y=reports, fill=state)) +
  geom_area(alpha = 0.6)


state.plot +
  theme(legend.position="bottom") +
  theme_bw() #+
```



ggplot 2

- Datacamp has 2 great module on plotting
- ggplot works by layers, like photoshop, You build it one layer at a time.

```
setwd("~/Dropbox/MethodsCamp/2018/Programming Lectures/Day2Programming/Assignment")
library(tidyverse)
dirty <- read.csv("stateyearreports.csv")
stateyear <- dirty %>% gather(state, reports, -c(1:2))

state.plot <- stateyear %>%
  ggplot(aes(x=year, y=reports, fill=state)) +
  geom_area(alpha = 0.6)


state.plot +
  theme(legend.position="bottom") +
  theme_bw() #+
```