Using broom and tidyr to format your data

long-form, wide-form, tidying, oh my!

Sara Weston and Debbie Yee

Psychological & Brain Sciences
Washington University in St. Louis

February 24, 2017

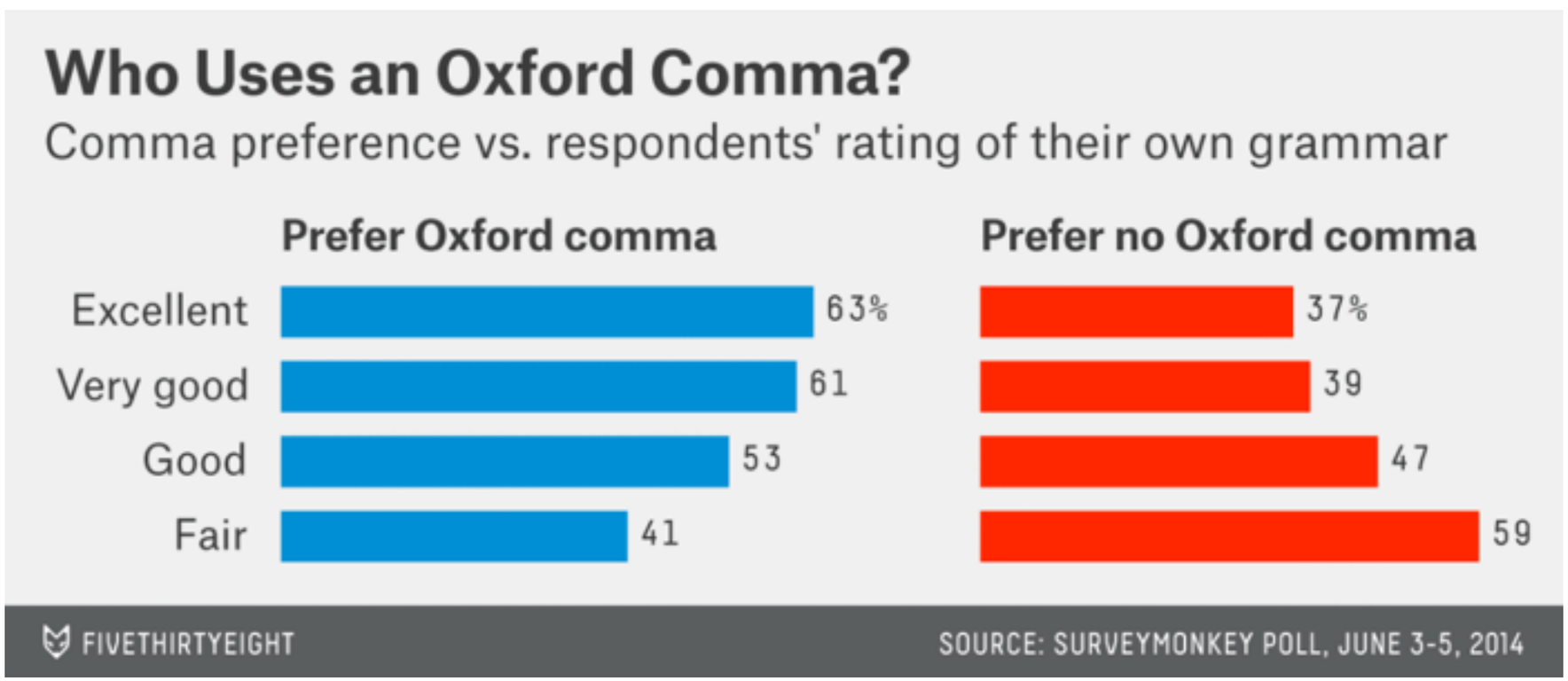
Today we will cover:

- **tidyr**: converting data frames from long-form to wide-form (and vice versa)
- **broom:** visualizing your statistical and regression model outputs
- **tibbles:** not your average table (e.g., a modern take on the classic data frame)

Oxford Comma Dataset

- Poll on the Oxford Comma: https://fivethirtyeight.com/datalab/elitist-superfluous-or-popular-we-polled-americans-on-the-oxford-comma/

- *"The people who tend to prefer the Oxford comma also tend to be the kind of people who will tell a survey that they think their own grammar is excellent."*



Load the dataset

```
library(fivethirtyeight)
data_oxford <- comma_survey
```

(Hint: Make sure that you've installed the fivethirtyeight package!)

comma_survey {fivethirtyeight} R Documentation

Elitist, Superfluous, Or Popular? We Polled Americans on the Oxford Comma

Description

The raw data behind the story "Elitist, Superfluous, Or Popular? We Polled Americans on the Oxford Comma" http://fivethirtyeight.com/datalab/elitist-superfluous-or-popular-we-polled-americans-on-the-oxford-comma/.

tidyr

- **tidyr** is a R package that makes it easy to “tidy” your data.
- **Two main functions or “verbs”:**
  - **gather()** is used to convert wide-form to long-form
  - **spread()** is used to convert long-form to wide-form
  - **separate()** & **unite()** are also useful (but less commonly used)
- ***NOTE 1:*** If you want to generally reshape or aggregate data, you’ll want to use the dplyr package
- ***NOTE 2:*** tidyr works well with dplyr data pipelines

gather(): wide-form to long-form

- gather() takes multiple columns and collapses them into key-value pairs. It has three arguments:
  - data = your data frame
  - key = explains what the information describes
  - value = actual information

key-value pair example

Password: ILoveR

key

value

gather(): wide-form to long-form

"wide form"

```
## Source: local data frame [6 x 4]
##
##       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
```

"long form"

```
## Source: local data frame [12 x 4]
##
##        country year        key     value
## 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
```

keys

values

first, some data formatting

Because I only care about the subject ID, and a few other variables, I'm going to select only the relevant columns:

```
data_oxford_20<-
dplyr::select(data_oxford,respondent_id,
heard_oxford_comma, data_singular_plural)
```

Also, this dataset is HUGE, so I'm going to work with the first 20 data points

```
data_oxford_20<-slice(data_oxford_20,c(1:20))
```

gather(): wide-form to long-form

```
data_oxford_long<-gather(data = data_oxford_20,
“question”,"answer",2:3)
```

```
> data_oxford_long
# A tibble: 40 × 3
   respondent_id              question answer
           <dbl>                 <chr>  <lgl>
1     3292644552   heard_oxford_comma  FALSE
2     3292644552 data_singular_plural   TRUE
3     3292648325   heard_oxford_comma  FALSE
4     3292648325 data_singular_plural   TRUE
5     3292653724   heard_oxford_comma   TRUE
6     3292653724 data_singular_plural  FALSE
7     3292692304   heard_oxford_comma   TRUE
8     3292692304 data_singular_plural  FALSE
9     3292702854   heard_oxford_comma   TRUE
10    3292702854 data_singular_plural   TRUE
# ... with 30 more rows
```

spread(): long-form to wide-form

- **spread()** distributes a pair of key-value columns into a field of cells. Thus, the “keys” become separate columns, making the data more “wide.”

- **spread()** takes three optional arguments addition to data, key, and value:

  - fill = if the combinations of valuables result in non-existent data, then puts an NA in the cell.

  - convert = if the value column contains different data types, convert will convert strings to doubles, integers, factors, etc.

  - drop = controls how spread() handles factors in the key column

spread(): long-form to wide-form

"wide form"

```
## Source: local data frame [6 x 4]
##
##       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
```

"long form"

```
## Source: local data frame [12 x 4]
##
##        country year        key     value
## 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
```

keys

values

spread(): long-form to wide-form

```
data_oxford_wide<-spread(data = data_oxford_long,
key = question, value = answer)
```

```
> data_oxford_wide
# A tibble: 20 × 3
   respondent_id data_singular_plural heard_oxford_comma
*          <dbl>                <lgl>              <lgl>
1     3292644552                 TRUE              FALSE
2     3292648325                 TRUE              FALSE
3     3292653724                FALSE               TRUE
4     3292692304                FALSE               TRUE
5     3292702854                 TRUE               TRUE
6     3292707770                 TRUE               TRUE
7     3292720964                 TRUE               TRUE
8     3292735069                 TRUE              FALSE
9     3292742681                FALSE              FALSE
10    3292753795                FALSE               TRUE
11    3292860428                 TRUE              FALSE
12    3292863455                FALSE               TRUE
```

Putting it together: creating a pipeline

```
data_oxford %>%
  dplyr::select(respondent_id,heard_oxford_comma,
  data_singular_plural) %>%
  slice(c(1:20)) %>%
  gather("question","answer",2:3) %>%
  arrange(respondent_id)
```

```
# A tibble: 40 x 3
   respondent_id             question answer
           <dbl>                <chr>  <lgl>
1     3292644552   heard_oxford_comma  FALSE
2     3292644552 data_singular_plural   TRUE
3     3292648325   heard_oxford_comma  FALSE
4     3292648325 data_singular_plural   TRUE
5     3292653724   heard_oxford_comma   TRUE
6     3292653724 data_singular_plural  FALSE
7     3292692304   heard_oxford_comma   TRUE
8     3292692304 data_singular_plural  FALSE
9     3292702854   heard_oxford_comma   TRUE
10    3292702854 data_singular_plural   TRUE
# ... with 30 more rows
>
```

More on pipelines:

- http://genomicsclass.github.io/book/pages/dplyr_tutorial.html

What about missing data?

- **drop_na()** = drops rows containing missing values
- **replace_na()** = replaces missing values

```
replace_na(data_oxford, list(gender="MISSING4EVER"))
```

```
> replace_na(data_oxford, list(gender = "MISSING4EVER") )
# A tibble: 1,129 x 13
   respondent_id       gender    age  household_income                        education           location
           <dbl>        <chr> <fctr>            <fctr>                           <fctr>              <chr>
1     3292953864         Male  30-44 $50,000 - $99,999                   Bachelor degree     South Atlantic
2     3292950324         Male  30-44 $50,000 - $99,999                   Graduate degree           Mountain
3     3292942669         Male  30-44                NA                                NA East North Central
4     3292932796         Male  18-29                NA     Less than high school degree    Middle Atlantic
5     3292932522 MISSING4EVER     NA                NA                                NA               <NA>
6     3292926586         Male  18-29 $25,000 - $49,999 Some college or Associate degree        New England
7     3292908135         Male  18-29       $0 - $24,999 Some college or Associate degree            Pacific
```

I was replaced!

tidyr vs.**reshape2**

- Many of you may have learned how to reshape your data using a package called “reshape2”

- Did you know that both packages are written by the same person? (Hadley Wickham). Therefore, you use just use the new package tidyr.

- **melt()** => **gather()**, **dcast()** => **spread()**, etc.

- https://www.r-bloggers.com/how-to-reshape-data-in-r-tidyr-vs-reshape2/

Some **tidyr** resources

- **Rstudio blog:** https://blog.rstudio.org/2014/07/22/introducing-tidyr/
- **tidyr vignette:** ftp://cran.r-project.org/pub/R/web/packages/tidyr/vignettes/tidy-data.html
- **a paper on tidyr:** http://vita.had.co.nz/papers/tidy-data.html
- **tutorial on tidyr:** http://garrettgman.github.io/tidying/

broom

- **broom** is an R package that takes the messy output of built-in functions in R (e.g., lm, nls, t.test) and turns them into tidy data frames.

- Why do we care?

- Well, model inputs usually require tidy inputs, but model outputs tend to be messy.

- broom solves this problem by making it easy to combine results from multiple models.

Iris dataset

- iris is a dataset pre-loaded into R

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
>
```

Let’s run a linear model on the iris dataset

```
model<-lm(formula = Petal.Width ~ Sepal.Width, data
= iris)
summary(model)
```

```
> summary(model)

Call:
lm(formula = Petal.Width ~ Sepal.Width, data = iris)

Residuals:
     Min       1Q   Median       3Q      Max
-1.38424 -0.60889 -0.03208  0.52691  1.64812

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.1569     0.4131   7.642 2.47e-12 ***
Sepal.Width  -0.6403     0.1338  -4.786 4.07e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7117 on 148 degrees of freedom
Multiple R-squared:  0.134,     Adjusted R-squared:  0.1282
F-statistic: 22.91 on 1 and 148 DF,  p-value: 4.073e-06
```

Messy model outputs!

- However, upon taking a closer look at the model output, the list of 12 “things” in the model are hard to extract.

- …Gross.

```
model        List of 12
    coefficients : Named num [1:2] 3.…
    coefficients : Named num [1:2] 3.16 -0.64 :2] "…
    residuals : Named num [1:150] -0.…
    ..- attr(*, "names")= chr [1:150]…
    effects : Named num [1:150] -14.6…
    ..- attr(*, "names")= chr [1:150]…
    rank : int 2
    fitted.values: Named num [1:150] …
    ..- attr(*, "names")= chr [1:150]…
    assign : int [1:2] 0 1
    qr :List of 5
    ..$ qr : num [1:150, 1:2] -12.247…
    .. ..- attr(*, "dimnames")=List o…
```

Tidy model outputs

- With the broom package, we can use the tidy() function to create a table of the model parameters, which is easier for us to extract!

- `model_tidy<_tidy(model)`

```
model_tidy 2 obs. of 5 variables
  term : chr "(Intercept)" "Sepal.W…
estimate : num 3.16 -0.64 3.16 -0.64
  std.error: num 0.413 0.134
  statistic: num 7.64 -4.79
  p.value : num 2.47e-12 4.07e-06
```

```
> model_tidy
         term   estimate std.error statistic      p.value
1 (Intercept)  3.1568723 0.4130820  7.642242 2.474053e-12
2 Sepal.Width -0.6402766 0.1337683 -4.786461 4.073229e-06
>
```

Some **broom** resources

- **broom vignette:** ftp://cran.r-project.org/pub/R/web/packages/broom/vignettes/broom.html
- **R bloggers:** https://www.r-bloggers.com/broom-a-package-for-tidying-statistical-models-into-data-frames/
- **More R bloggers:** https://www.r-bloggers.com/slides-from-my-talk-on-the-broom-package/
- **From the cran:** https://cran.r-project.org/web/packages/broom/broom.pdf

tibbles: a modern take on the classic data frame

- Once upon a time, R (and S) best took data in the format of data frames. But then data became exponentially bigger and developers decided that a new type of data frame needed to be invented. Thus, the “tibble” was born.

- Advantages of using tibbles:
  - when you print the tibble, only see first 10 rows
  - when you subset, you always output the same format (a subset of a tibble will always be a tibble)

- as_tibble() can be used to transform a data frame into a tibble

- Basically, a tibble is a more user friendly version of a data frame. To use them, you need to load the “tibble” package.

Some tibble resources

- **tibble vignette:** https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html

- **Statistical Tools:** http://www.sthda.com/english/wiki/tibble-data-format-in-r-best-and-modern-way-to-work-with-your-data

- **tibbles in R for Data Science:** http://r4ds.had.co.nz/tibbles.html#exercises-18

more in the tidyverse

- **tidyr** and **broom** are just two packages in the tidyverse. for more fun existing and new packages for data manipulation, check out the rstudio blog: https://blog.rstudio.org/category/tidyverse/

- A few examples:
  ggplot2
  dplyr
  tidyr
  ready
  purr
  tibble

Thanks for your attention!

Any Questions? Email Sara or Debbie.

Also, check out more of “R” tutorials here:
https://debyeeneuro.com/r-tutorials/