Using broom and tidyr to format your data

Debbie Yee and Sara Weston
R Workshop

Department of Psychological and Brain Sciences
Washington University in St. Louis

February 24, 2017

Have you ever wanted to find an easy way to transform your data from wide-form to long-form (or vice versa), but just got frustrated with R? Or maybe youve run lots of regressions and want to extract your coefficients, standard errors, or statistics in an easy way, but didnt know how. Or perhaps you want to know what the heck a tibble is. Never fear! This R tutorial will cover two packages (broom and tidyr) that we believe can help you on your quest to format your data frames and or make easily accessible the outputs from your statistical tests. Additionally, we will discuss what a "tibble" is.

(As an extra plug, if youve been a fan of our favorite data manipulation package dplyr, you will be guaranteed to loved broom and tidyr, since they are written by the same folks).

1 Load required packages

If run into an error, check if you have these packages already installed!

```
library(fivethirtyeight) # for this tutorial's dataset
library(tidyr)
library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##    filter, lag
## The following objects are masked from 'package:base':
##
##    intersect, setdiff, setequal, union

library(broom)
library(tibble)
```

2 Oxford Comma Dataset

For the dataset, we will be using the raw data from a survey conducted on Americans about their opinion on the highly controversial oxford comma. The conclusion from the poll is the following: *"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."*

The full story behind the dataset can be found here: https://fivethirtyeight.com/datalab/elitist-superfluous-or-popular-we-polled-americans-on-the-oxford-comma/

```
# load in the data. make sure you have the fivethirtyeight package loaded
data_oxford <- comma_survey

# looking at the first few columns and rows of the dataset
head(data_oxford)
```

```
## # A tibble: 6  13
##   respondent_id gender    age  household_income
##           <dbl>  <chr> <fctr>            <fctr>
## 1    3292953864   Male  30-44 $50,000 - $99,999
## 2    3292950324   Male  30-44 $50,000 - $99,999
## 3    3292942669   Male  30-44                NA
## 4    3292932796   Male  18-29                NA
## 5    3292932522   <NA>     NA                NA
## 6    3292926586   Male  18-29 $25,000 - $49,999
## # ... with 9 more variables: education <fctr>, location <chr>,
## #   more_grammar_correct <chr>, heard_oxford_comma <lgl>,
## #   care_oxford_comma <fctr>, write_following <chr>,
## #   data_singular_plural <lgl>, care_data <fctr>,
## #   care_proper_grammar <fctr>
```

```
# Because I only care about the subject ID, and a few other variables,
# im going to select only the relevant columns:
# column 1 = participant ID
# column 2 = heard of oxford comma (boolean, TRUE or FALSE)
# column 3 = data_singular_pluar (boolean, TRUE or FALSE)
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 datapoints
data_oxford_20<-slice(data_oxford_20,c(1:20))
```

3 tidyr

tidyr is an R package written by Hadley Wickham (http://hadley.nz/) that makes it easy to"tidy" your data. It is fairly limited in scope (it mostly just converts data frames from long-form to wide-form and vice versa), it works well with other tidyverse packages (e.g., dplyr) to manipulate your data.

3.1 gather: converting data frames from wide form to long form

Our original dataset data_oxford_20 is actually in wide format. With the gather() function, we can consolidate the key-value pairs into a single column. We can see that once we have converted the dataframe to long form, the key-value pairs provide the same information in a different format.

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

3.2 spread: converting data frames from long form to wide form

Conversely, if we wanted to switch from long form to wide form, we would use the spread() function.

```
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
## 13    3292869879                FALSE                TRUE
## 14    3292908135                 TRUE                TRUE
## 15    3292926586                 TRUE               FALSE
## 16    3292932522                FALSE               FALSE
## 17    3292932796                FALSE                TRUE
## 18    3292942669                 TRUE                TRUE
## 19    3292950324                FALSE               FALSE
## 20    3292953864                FALSE                TRUE
```

3.3 Putting it together: creating a pipeline

Here, we can combine all of the individual dplyr and tidyr functions into a pipeline. This is an example of how I might want to manipulate my data into long-form. For more extensive review of dplyr and pipelines, check out this link: http://genomicsclass.github.io/book/pages/dplyr_tutorial.html

```
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  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
```

3.4 What about missing data?

When you are missing data, you can use various functions to either drop or replace your NAs

```
# dropping rows with NA values
drop_na(data_oxford)
```

```
## # A tibble: 825  13
##    respondent_id gender    age  household_income
##            <dbl>  <chr> <fctr>            <fctr>
## 1     3292953864   Male  30-44 $50,000 - $99,999
## 2     3292950324   Male  30-44 $50,000 - $99,999
## 3     3292926586   Male  18-29 $25,000 - $49,999
## 4     3292908135   Male  18-29      $0 - $24,999
## 5     3292869879   Male  18-29 $25,000 - $49,999
## 6     3292863455   Male  30-44 $50,000 - $99,999
## 7     3292860428   Male  30-44         $150,000+
## 8     3292753795   Male  18-29 $50,000 - $99,999
## 9     3292742681   Male  30-44 $25,000 - $49,999
## 10    3292735069   Male  18-29 $50,000 - $99,999
## # ... with 815 more rows, and 9 more variables: education <fctr>,
## #   location <chr>, more_grammar_correct <chr>, heard_oxford_comma <lgl>,
## #   care_oxford_comma <fctr>, write_following <chr>,
## #   data_singular_plural <lgl>, care_data <fctr>,
## #   care_proper_grammar <fctr>
```

```
# replacing your NA values with another value or string.
replace_na(data_oxford, list(gender = "MISSING4EVER"))
```

```
## # A tibble: 1,129  13
##    respondent_id       gender    age  household_income
##            <dbl>        <chr> <fctr>            <fctr>
## 1     3292953864         Male  30-44 $50,000 - $99,999
## 2     3292950324         Male  30-44 $50,000 - $99,999
## 3     3292942669         Male  30-44                NA
## 4     3292932796         Male  18-29                NA
## 5     3292932522 MISSING4EVER     NA                NA
## 6     3292926586         Male  18-29 $25,000 - $49,999
## 7     3292908135         Male  18-29      $0 - $24,999
## 8     3292869879         Male  18-29 $25,000 - $49,999
## 9     3292863455         Male  30-44 $50,000 - $99,999
## 10    3292860428         Male  30-44         $150,000+
## # ... with 1,119 more rows, and 9 more variables: education <fctr>,
## #   location <chr>, more_grammar_correct <chr>, heard_oxford_comma <lgl>,
## #   care_oxford_comma <fctr>, write_following <chr>,
## #   data_singular_plural <lgl>, care_data <fctr>,
## #   care_proper_grammar <fctr>
```

4 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.This is convenient, since it makes the messy outputs from statistical models in a tidy format, which we can use to combine results from multiple models.

4.1 iris dataset

Back to the iris dataset. This is a default dataset in 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
```

4.2 A linear model on the iris dataset

Next, we run a linear regression with Petal with predicted by sepal width and Petal Length. As we can see by the output of the model reveals that both sepal width and petal length are significant predictors of petal width.

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

4.3 Using broom to tidy the model output

What if we want to extract the coefficients of the model? We can use the tidy() function from the broom package to clean up the messy output from our linear model. What's most cool, however, is that we can eventually combine the outputs of different models for comparison! (not shown here)

```
model_tidy<-tidy(model)
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
```

Questions? More Tutorials? Check our list of tutorials here: https://debyeeneuro.com/r-tutorials/