Data Manipulation with Tidyr

Kyle Hwang

Introduction:

In conducting any type of data analysis, cleaning your data is essential. We have already explored the *dplyr* package for data manipulation in this class, but we have not explored the functions within the *tidyr* package. These are very useful in transforming raw data to clean, tidy data.

My motivation for this post is that I wanted to learn about other data cleaning functions. Here are four functions that can help reshape the layout of your data:

- *spread()*: spreads rows into columns
- *gather()*: gathers columns into rows
- *unite()*: unite multiple columns into one
- *separate()*: separates multiple columns into one

These functions are used to tidy-up messy data, which is essential in data cleaning for data science applications. So far, the data sets we have used are all considered tidy data, which is critical for data manipulation, modeling, and visualization. However, you never know what sorts of messy raw data you may encounter, so we will explore tools you can use to tidy data yourself.

In order to use any package, we must first install it in R (only need to do this once) and load it with the function *library()*. We will also load readr to read in sample data sets I created.

```
#install.packages("tidyr")
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.4.2
```

```
library(readr)
```

What are key-value pairs and tidy data?

Before we dive into *spread()* and *gather()*, we should discuss data points in the context of pairs of a *key* and a *value*. The 'key' categorizes and describes what type of information is presented, whereas the 'value' is the information itself. For example, 'Course' is the key below and 'Stat 133' is the value.

```
'Course Stat 133'
```

```
## [1] "Course Stat 133"
```

Key and value pairs are not very useful alone because it is hard to relate multiple key and value pairs with each other if they correspond to one data point. This is why data tables are so useful because they organize key and value pairs into separate columns, where each row corresponds to one data point. However, this is only the case when we are looking at clean, tidy data which we are used to seeing in this course. Now we will explore untidy data, and four functions within the tidyr package.

Tidy data has columns has keys (variables) and rows as values (observations). Untidy data could have multiple keys in one column. For example:

```
#Reading in and displaying sample1 data for spread example
sample1 <- read_csv("sample1.csv")
```

```
## Parsed with column specification:
## cols(
##   country = col_character(),
##   year = col_integer(),
##   key = col_character(),
##   value = col_integer()
## )
```

```
sample1 <- sample1[1:12,]
sample1
```

```
## # A tibble: 12 x 4
##    country  year        key     value
##      <chr> <int>      <chr>     <int>
##  1     USA  2016     births    134132
##  2     USA  2016 population 321849485
##  3     USA  2017     births    500388
##  4     USA  2017 population 322349873
##  5  RUSSIA  2016     births    245098
##  6  RUSSIA  2016 population 144000000
##  7  RUSSIA  2017     births    342003
##  8  RUSSIA  2017 population 144342003
##  9  CANADA  2016     births    324934
## 10  CANADA  2016 population  36029590
## 11  CANADA  2017     births   1205393
## 12  CANADA  2017 population  37234983
```

spread(): spreads rows into columns

Spread basically turns rows into columns, converting the data from wide to long format. Pass into the function the name of the data frame, the name of the key column, and the name of the value column. You do not need quotes when referencing column names. Applying *spread()* to the *sample1* data from above,

```
#using the spread function to create 'tidy' data
spread(sample1, key, value)
```

```
## # A tibble: 6 x 4
##   country  year  births population
## *   <chr> <int>   <int>      <int>
## 1  CANADA  2016  324934   36029590
## 2  CANADA  2017 1205393   37234983
## 3  RUSSIA  2016  245098  144000000
## 4  RUSSIA  2017  342003  144342003
## 5     USA  2016  134132  321849485
## 6     USA  2017  500388  322349873
```

The function returns the data set with the key and value columns removed, and adds a column for each unique value, with the value becoming the column names of the new columns. The function maintains the relationships in the original data, and the output is now in a tidy format.

gather(): gathers columns into rows

Gather is the oppposite of spread() and gathers columns to collapse into rows. Here is our second sample data set.

```
#Reading in and displaying sample2 data for gather example
sample2 <- read_csv("sample2.csv")
```

```
## Warning: Missing column names filled in: 'X4' [4]
```

```
## Parsed with column specification:
## cols(
##   country = col_character(),
##   `2016` = col_integer(),
##   `2017` = col_integer(),
##   X4 = col_character()
## )
```

```
sample2 <- sample2[1:3, 1:3]
sample2
```

```
## # A tibble: 3 x 3
##   country    `2016`    `2017`
##     <chr>     <int>     <int>
## 1     USA 321849485 322349873
## 2  RUSSIA 144000000 144342003
## 3  CANADA  36029590  37234983
```

Pass in a data frame, a character string for the “key” column that it will make, a character string for the “value” column that it will make, and specify which columns to collapse into the key value pair (here with integer notation)

```
#exemplifying the gather function
gather(sample2, "year", "population", 2:3)
```

```
## # A tibble: 6 x 3
##   country  year population
##     <chr> <chr>      <int>
## 1     USA  2016  321849485
## 2  RUSSIA  2016  144000000
## 3  CANADA  2016   36029590
## 4     USA  2017  322349873
## 5  RUSSIA  2017  144342003
## 6  CANADA  2017   37234983
```

The function returns the data with the specified columns replaced with two new columns of a “key” column that contains the former column names of the removed columns, and a “value” column that contains the former values of the removed columns.

separate(): separates multiple columns into one

Separate turns a single character column into multiple columns by splitting the values of the column wherever a separator character appears.

```
#Reading in and displaying sample3 data to exemplify separate
sample3 <- read_csv("sample3.csv")
```

```
## Warning: Missing column names filled in: 'X4' [4]
## Parsed with column specification:
## cols(
##   country = col_character(),
##   year = col_integer(),
##   `birth rate` = col_character(),
##   X4 = col_character()
## )
```

```
sample3 <- sample3[1:6, 1:3]
sample3
```

```
## # A tibble: 6 x 3
##   country  year     `birth rate`
##     <chr> <int>            <chr>
## 1     USA  2016 134132/321849485
## 2     USA  2017 500388/322349873
## 3  RUSSIA  2016 245098/144000000
## 4  RUSSIA  2017 342003/144342003
## 5  CANADA  2016  324934/36029590
## 6  CANADA  2017 1205393/37234983
```

Pass in the name of the data set, the column to separate, a vector of strings to use as new column names. By default, values will be split wherever a character that is not a number or a letter appears. Non-alphanumeric characters are characters that are neither a number nor a letter. You can also specify a separator character by passing in the character to the sep argument.

```
#Creating a sample4 data set using the separate function to use in subsequent example
sample4 <- separate(sample3, 3, c("births", "population"), sep = "/")
sample4
```

```
## # A tibble: 6 x 4
##   country  year  births population
## *   <chr> <int>   <chr>      <chr>
## 1     USA  2016  134132  321849485
## 2     USA  2017  500388  322349873
## 3  RUSSIA  2016  245098  144000000
## 4  RUSSIA  2017  342003  144342003
## 5  CANADA  2016  324934   36029590
## 6  CANADA  2017 1205393   37234983
```

The output will be the data set with the specified column removed, and the values of the original column will be split by the character separator across new columns.

unite(): unite multiple columns into one

Unite is the opposite of the separate function, and combines multiple columns into one. In this example, we will unite the previous sample data together.

Pass in the name of the data set to reshape, the name of the new column to create, and the names of the columns to unite. Unite() will by default use an underscore to connect values from the separate columns.

```
#Exemplifying the unite function using the sample4 data set created above
unite(sample4, "birth rate", births, population)
```

```
## # A tibble: 6 x 3
##   country  year     `birth rate`
## *   <chr> <int>            <chr>
## 1     USA  2016 134132_321849485
## 2     USA  2017 500388_322349873
## 3  RUSSIA  2016 245098_144000000
## 4  RUSSIA  2017 342003_144342003
## 5  CANADA  2016  324934_36029590
## 6  CANADA  2017 1205393_37234983
```

You can pass in a specific separator with character string.

```
#Use of the 'sep' argument
unite(sample4, "birth rate", births, population, sep = "/")
```

```
## # A tibble: 6 x 3
##   country  year     `birth rate`
## *   <chr> <int>            <chr>
## 1     USA  2016 134132/321849485
## 2     USA  2017 500388/322349873
## 3  RUSSIA  2016 245098/144000000
## 4  RUSSIA  2017 342003/144342003
## 5  CANADA  2016  324934/36029590
## 6  CANADA  2017 1205393/37234983
```

The output is a copy of the data set that includes the new column, with the specified columns combined. We got back the sample3 data set we applied *separate* to earlier! If you would like to keep the original columns, add the argument remove = FALSE.

```
#Use of the 'remove' argument
unite(sample4, "birth rate", births, population, sep = "/", remove = FALSE)
```

```
## # A tibble: 6 x 5
##   country  year      `birth rate`  births population
## *   <chr> <int>             <chr>   <chr>      <chr>
## 1     USA  2016 134132/321849485  134132  321849485
## 2     USA  2017 500388/322349873  500388  322349873
## 3  RUSSIA  2016 245098/144000000  245098  144000000
## 4  RUSSIA  2017 342003/144342003  342003  144342003
## 5  CANADA  2016  324934/36029590  324934   36029590
## 6  CANADA  2017 1205393/37234983 1205393   37234983
```

Concluding Take Aways: "Tidy" Up Your Data:

We have now learned about the *tidyr* package with four useful functions: *spread, gather, separate, unite*. If you noticed, spread is the complement function of gather, and separate is the complement function of unite. We learned how these functions can make help make your data "tidy" with columns that represent variables and rows that represent values. Once data is formatted this way, you can apply dplyr functions and other data manipulation tools like we have done so in this course so far. Hope you learned about another aspect of data cleaning that we have not covered in this course so far!

References:

1. Data Manipulation with Tidyr: https://www.r-bloggers.com/data-manipulation-with-tidyr/
2. Tidyr Introduction: https://blog.rstudio.com/2014/07/22/introducing-tidyr/
3. R Code Chunks: http://rmarkdown.rstudio.com/authoring_rcodechunks.html
4. Tidying: http://garrettgman.github.io/tidying/
5. Reshaping Data: http://www.sthda.com/english/wiki/tidyr-crucial-step-reshaping-data-with-r-for-easier-analyses
6. Tidy Verse: http://tidyr.tidyverse.org/
7. Tidy Data Wikipedia: https://en.wikipedia.org/wiki/Tidy_data