Tidy data using 'tidyr'

Stat 133: Post 1

Mindy Lee

10/17/2017

Introduction

"This is not what I meant when I said 'we need better data cleansing!'"

www.iwaysoftware.com/go/dataquality

Of course it's not!

Data cleansing is a process of modifying, organizing, and correcting messy data into concise, relevant, and accurate data. Manipulating data is an important step of data analysis. We need a clean data in order to visualize the information and make statistical inference. In class we learned how to use 'dplyr' to organize and clean our data. In this post, I am going to introduce another package, 'tidyr', to use along with 'dplyr'. I will go over the basic funtions of 'tidyr' and provide some examples on how to combine 'tidyr' with what we already know about 'dplyr' and 'ggplot2'.

```
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(tidyr)
library(ggplot2)
```

What is ‘tidyr’?

‘tidyr’ is a package developed by the creator of ‘dplyr’ and ‘ggplot2’, Hadley Wickham. ‘tidyr’ treats every row as an observaton and every column as a variable, this makes tidying your data easy as you have a consistent way of referring to your observations and variables.

Functions of tidyr

‘tidyr’ has 4 main functions we can use to manipulate our data: gather, spread, separate, and unite.

Before I introduce each of these functions, I’m going to create a simple data frame “bears”" consisting of 5 UC Berkeley students with their major GPA, cumulative GPA, and birth year & month. This data frame will be used to illustrate ‘tidyr’ functions.

```
bears <- data.frame(
  names = c("claire", "wilson", "jael", "angus", "gary"),
  major = c(3.7, 3.3, 3.3, 3.0, 2.6),
  cum = c(3.5, 3.2, 3.5, 2.7, 2.8),
  yr_mo = c("1997.apr", "1993.jun", "1995.oct", "1995.may", "1996.jun"))
bears
```

```
##    names major cum    yr_mo
## 1 claire   3.7 3.5 1997.apr
## 2 wilson   3.3 3.2 1993.jun
## 3   jael   3.3 3.5 1995.oct
## 4  angus   3.0 2.7 1995.may
## 5   gary   2.6 2.8 1996.jun
```

gather()

gather() takes multiple columns and gather them under one key. It turns “wide” data longer.

For instance, I want to have all the gpa, regardless of the types, to be in the same column. I will use the function gather().

```
long_bears <- bears %>% gather(gpa_type, gpa, major:cum)
long_bears
```

```
##     names    yr_mo gpa_type gpa
## 1  claire 1997.apr    major 3.7
## 2  wilson 1993.jun    major 3.3
## 3    jael 1995.oct    major 3.3
## 4   angus 1995.may    major 3.0
## 5    gary 1996.jun    major 2.6
## 6  claire 1997.apr      cum 3.5
## 7  wilson 1993.jun      cum 3.2
## 8    jael 1995.oct      cum 3.5
## 9   angus 1995.may      cum 2.7
## 10   gary 1996.jun      cum 2.8
```

spread()

From its name, spread() does the opposite of what gather() does. spread() takes a keyvalue pair and spread them into multiple columns, turning “long” data to wider data.

We can use spread() to convert the new data frame (“long_bears”) we created using gather() back to the original data frame.

```
long_bears %>% spread(gpa_type, gpa)
```

```
##    names    yr_mo cum major
## 1  angus 1995.may 2.7   3.0
## 2 claire 1997.apr 3.5   3.7
## 3   gary 1996.jun 2.8   2.6
## 4   jael 1995.oct 3.5   3.3
## 5 wilson 1993.jun 3.2   3.3
```

separate()

separate() splits a single variable into two.

Let’s say that I find student’s birth year and month hard to read as one variable, and I want to separate them into two different columns. I will turn to the function separate().

```
sep_bears <- bears %>% separate(yr_mo, c("year", "month"))
sep_bears
```

```
##    names major cum year month
## 1 claire   3.7 3.5 1997   apr
## 2 wilson   3.3 3.2 1993   jun
## 3   jael   3.3 3.5 1995   oct
## 4  angus   3.0 2.7 1995   may
## 5   gary   2.6 2.8 1996   jun
```

unite()

unite() merges two variables into one, undoing what separate() does. It combines two variables of a single observation into just one variable.

Let's revert "sep_bears" back to bears using unite().

```
sep_bears %>% unite(yr_mo, year, month, sep = ".")
```

```
##    names major cum    yr_mo
## 1 claire   3.7 3.5 1997.apr
## 2 wilson   3.3 3.2 1993.jun
## 3   jael   3.3 3.5 1995.oct
## 4  angus   3.0 2.7 1995.may
## 5   gary   2.6 2.8 1996.jun
```

Using 'tidyr' with 'dplyr'

Now that we have learned the basic functions of 'tidyr', let's see how 'tidyr' can be used with 'dplyr'.

Here is a raw data set of Women's High Jump World Records (as of Sep 2016).

```
women <- read.csv("../data/women.csv")
head(women, 5)
```

```
##                     Height                  Athlete          Date
## 1   1.46 m (4 ft 9 1⁄2 in)     Nancy Voorhees (USA)   20 May 1922
## 2 1.485 m (4 ft 10 1⁄2 in)    Elizabeth Stine (USA)   26 May 1923
## 3 1.485 m (4 ft 10 1⁄2 in) Sophie Eliott-Lynn (GBR) 6 August 1923
## 4      1.524 m (5 ft 0 in)      Phyllis Green (GBR)  11 July 1925
## 5  1.552 m (5 ft 1 1⁄8 in)      Phyllis Green (GBR) 2 August 1926
##          Place
## 1  Simsbury[1]
## 2    Leonia[1]
## 3 Brentwood[1]
## 4    London[1]
## 5    London[1]
```

Example 1

Let's say I want to look at the trend of how the high jump world record is renewed (comparing height in meters and year), with the atheletes grouped by their nationality. We can use 'dplyr' and 'tidyr' to clean up the data.

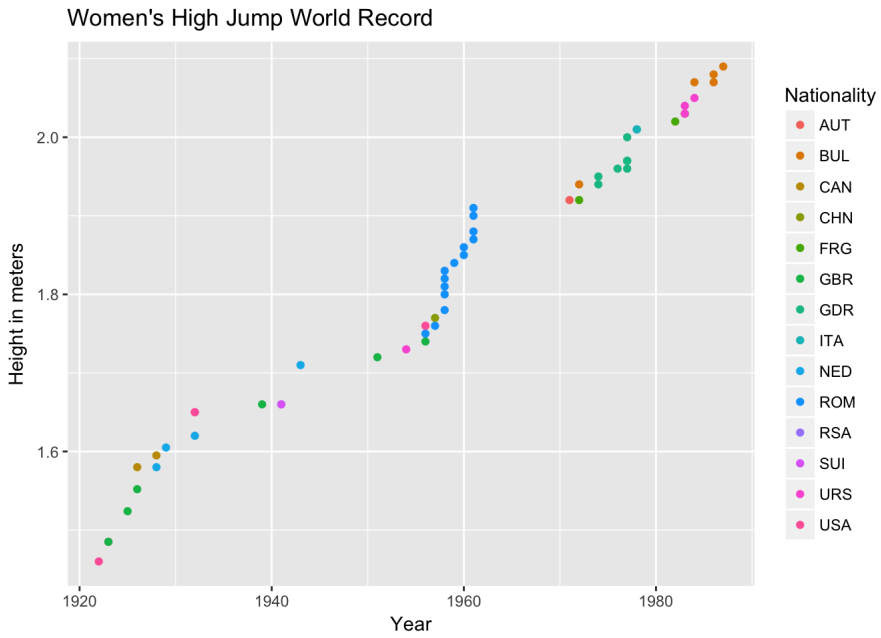
```
clean_women <- women %>%
        # split up height in meters and height in ft & in
        separate(Height, c("Height_m", "Height (ft & in)"), sep = "[m]", convert = TRUE) %>%
        # split up date into Day, Month, and Year. Convert Day and Year to integers
        separate(Date, c("Day", "Month", "Year"), sep = " ", convert = TRUE) %>%
        # split up Athlete into athlete names and nationality
        # drop the extra variables to make each row equal length
        separate(Athlete, c("Athlete", "Nationality"), sep = "[()]", extra = "drop") %>%
        # select the columns with desired info
        select("Height_m", "Athlete", "Nationality", "Day", "Month", "Year", "Place")

head(clean_women, 5)
```

```
##   Height_m             Athlete Nationality Day  Month Year        Place
## 1    1.460     Nancy Voorhees          USA  20    May 1922  Simsbury[1]
## 2    1.485    Elizabeth Stine          USA  26    May 1923    Leonia[1]
## 3    1.485 Sophie Eliott-Lynn          GBR   6 August 1923 Brentwood[1]
## 4    1.524      Phyllis Green          GBR  11   July 1925    London[1]
## 5    1.552      Phyllis Green          GBR   2 August 1926    London[1]
```

We can visualize it using ggplot2.

```
ggplot(clean_women, aes(x = Year, y = Height_m)) +
  geom_point(aes(color = Nationality)) +
  ggtitle(label = "Women's High Jump World Record") +
  ylab("Height in meters")
```



Above we can see the graph shows the recorded height over the years and the color of the dots represent the nationalities of the athletes that broke the records. Notice that the trend ended before 1990, the record hasn't been renewed for 30 years!

Example 2

Let's say I want to know who, if anyone, broke the record at the 1972 Olympics in Munich. Instead of using a two part filter(), I can use unite to combine Year and Place and filter the Event.

```
clean_women %>%
  unite(Event, Year, Place) %>%
  filter(Event == "1972_Munich[1]") %>%
  select("Athlete")
```

```
##            Athlete
## 1 Ulrike Meyfarth
```

Example 3

I want to know in what years athletes broke the record in London and Rome. You can use spread() to display the Year record broke in London and Rome in one chunk of commands.

```
clean_women %>%
  filter(Place == "London[1]" | Place == "Rome[1]") %>%
  spread(Place, Year) %>%
  select("London[1]", "Rome[1]")
```

```
##   London[1] Rome[1]
## 1      1925      NA
## 2      1926      NA
## 3      1951      NA
## 4        NA    1974
## 5      1983      NA
## 6      1983      NA
## 7        NA    1987
```

Example 4

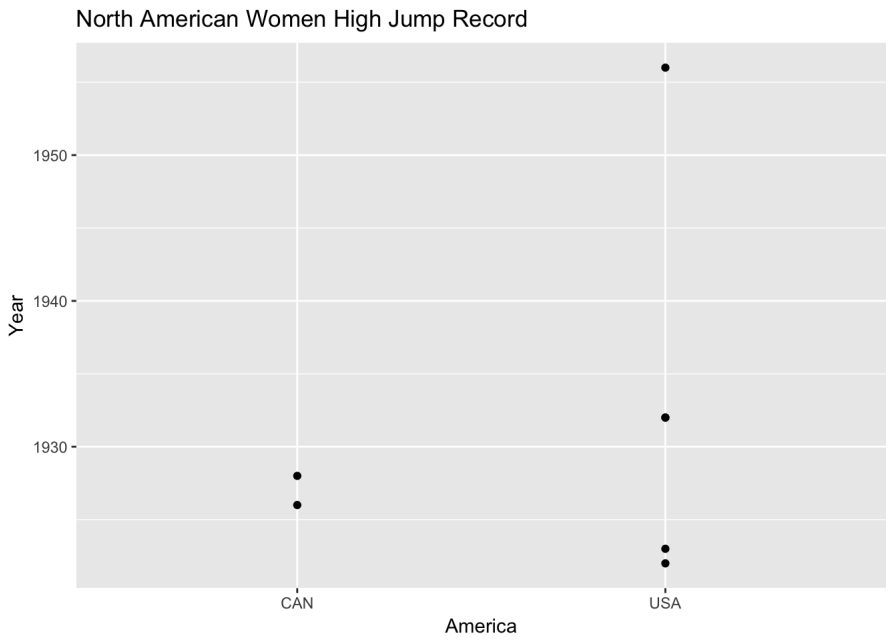
I want to know how many times and in what years have North Americans (United States and Canada) broken the records.

```
america <- clean_women %>%
    filter(Nationality == "USA" | Nationality == "CAN") %>%
    # spread the data based on Nationality
    spread(Nationality, Year) %>%
    select("USA", "CAN") %>%
    # gather USA and CAN under one key
    gather(America, Year) %>%
    # remove NAs
    filter(!is.na(Year))
america
```

```
##   America Year
## 1     USA 1922
## 2     USA 1923
## 3     USA 1932
## 4     USA 1932
## 5     USA 1956
## 6     CAN 1926
## 7     CAN 1928
```

Visualize using ggplot2.

```
ggplot(america, aes(x = America, y = Year)) +
  geom_point() +
  ggtitle(label = "North American Women High Jump Record")
```



From the graph, we can see that North Americans broke the women high jump record 6 times. Canada broke the record twice back in the 1920s. United States broke the record 4 times with the most recent one in the 1950s.

In summary…

'tidyr' is a simple and efficient package that helps you tidy your data. Along with 'dplyr' you can manipulate your data at ease. The four main functions of 'tidyr': gather(), spread(), separate(), and unite(), are especially handy when you are dealing with raw data. You can easily break down and regroup the data based on your needs.

I hope you enjoyed this post and learned something from it! Thank you!

References

- https://cran.r-project.org/web/packages/tidyr/index.html
- https://blog.rstudio.com/2014/07/22/introducing-tidyr/
- https://rpubs.com/bradleyboehmke/data_wrangling
- https://sesync-ci.github.io/data-manipulation-in-R-lesson/2016/07/26/
- http://tclavelle.github.io/dplyr-tidyr-tutorial/
- http://strimas.com/r/tidy-sf/
- http://garrettgman.github.io/tidying/