Data Wrangling Part 1

Welcome to SummeR of R at the Brandeis Library!

Margarita Corral

mcorral@brandeis.edu

Make an appointment with Margarita!

Shannon Hagerty

shannonhagerty@brandeis.edu

Make an appointment with Shannon!

Data Wrangling Part 1: tidyr & dplyr

Today we're going to be focused on getting data into a tidy format and some of the core functions in the tidyverse to transform your data! Our focus is going to be on the *tidyr* and *dplyr* packages in the tidyverse.

A good guide for this material is the Data Wrangling Cheat Sheet by RStudio

We start with loading the tidyverse (if you are new to the series you need to install the package first, delete the hashtag at line 9 to do that)

```
#install.packages('tidyverse')
library(tidyverse)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## -- Attaching packages ------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.2     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ---------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The *tidyr* package has two main functions that are meant to help convert your data into a tidy format.

1. gather() We use gather when we have multiple observations in one row (as opposed to tidy format with one observation in a row)

Remember our Game of Thrones Untidy dataset?

```
Untidy_GOT<-read_csv("UntidyGOT.csv") #read in the dataframe
```

```
## Parsed with column specification:
## cols(
##   Survives = col_character(),
```

```
##   Dies = col_character()
## )
```

```
print(Untidy_GOT)
```

```
## # A tibble: 5 x 2
##   Survives      Dies
##   <chr>         <chr>
## 1 Jon Snow      Daenerys
## 2 Arya          The Mountain
## 3 Sansa         Brienne
## 4 Samwell Tarly <NA>
## 5 Gilly         <NA>
```

We can convert this to a tidy format by using the gather function:

```
Tidy_GOT <- gather(Untidy_GOT, Survives, Dies, key='Outcome', value='Character', na.rm=TRUE)
print(Tidy_GOT)
```

```
## # A tibble: 8 x 2
##   Outcome  Character
##   <chr>    <chr>
## 1 Survives Jon Snow
## 2 Survives Arya
## 3 Survives Sansa
## 4 Survives Samwell Tarly
## 5 Survives Gilly
## 6 Dies     Daenerys
## 7 Dies     The Mountain
## 8 Dies     Brienne
```

2. spread() We use spread when we have multiple rows for one observation. Here's an example:

```
UntidyHeightWeight <- read_csv('UntidyHeightWeight.csv')
```

```
## Parsed with column specification:
## cols(
##   Person = col_character(),
##   Measurement = col_character(),
##   Value = col_double()
## )
```

```
print(UntidyHeightWeight)
```

```
## # A tibble: 8 x 3
##   Person Measurement Value
##   <chr>  <chr>       <dbl>
## 1 A      Height         60
## 2 A      Weight        120
## 3 B      Height         75
## 4 B      Weight        200
## 5 C      Height         68
## 6 C      Weight        170
## 7 D      Height         40
## 8 D      Weight         60
```

The key argument tells you which column in the untidy dataframe you want to be converted into column names in the tidy version, and the value is the column you want to be inside the cells of the dataframe.

```
TidyHeightWeight <- spread(UntidyHeightWeight, key=Measurement, value=Value)
print(TidyHeightWeight)
```

```
## # A tibble: 4 x 3
##   Person Height Weight
##   <chr>   <dbl>  <dbl>
## 1 A          60    120
## 2 B          75    200
## 3 C          68    170
## 4 D          40     60
```

There are 5 Main *dplyr* functions, and you already know some of them!

1. filter(): gets specific rows from a dataframe

2. select(): lets us pull out specific columns from a dataframe

3. summarize(): allows you to create summary statistics from columns in a dataframe.

4. arrange(): diplays the dataframe sequentially based on variables.

```
arrange(TidyHeightWeight, Height)
```

```
## # A tibble: 4 x 3
##   Person Height Weight
##   <chr>   <dbl>  <dbl>
## 1 D          40     60
## 2 A          60    120
## 3 C          68    170
## 4 B          75    200
```

```
arrange(TidyHeightWeight, desc(Weight))
```

```
## # A tibble: 4 x 3
##   Person Height Weight
##   <chr>   <dbl>  <dbl>
## 1 B          75    200
## 2 C          68    170
## 3 A          60    120
## 4 D          40     60
```

4. mutate(): Let's you create a new column!

```
Tidy_BMI<-mutate(TidyHeightWeight, BMI = 703*Weight/ (Height)^2)
print(Tidy_BMI)
```

```
## # A tibble: 4 x 4
##   Person Height Weight   BMI
##   <chr>   <dbl>  <dbl> <dbl>
## 1 A          60    120  23.4
## 2 B          75    200  25.0
## 3 C          68    170  25.8
## 4 D          40     60  26.4
```

The five main *dplyr* functions have similarities:

1. all work with `group_by()`
2. Need the dataframe name as their first argument inside the parentheses

3. Have additional arguments that describe what to do with dataframe

4. Have a new dataframe as output

- Summarized from Ch. 3 in R for Data Science

My favorite dplyr function

case_when() : Lets you do different things based on a condition

```
Tidy_BMI<-mutate(Tidy_BMI, Height_Class = case_when( Height > 65 ~ 'Tall',
                                                     Height < 65 ~ 'Short'))
print(Tidy_BMI)
```

```
## # A tibble: 4 x 5
##   Person Height Weight   BMI Height_Class
##   <chr>   <dbl>  <dbl> <dbl> <chr>
## 1 A          60    120  23.4 Short
## 2 B          75    200  25.0 Tall
## 3 C          68    170  25.8 Tall
## 4 D          40     60  26.4 Short
```

Let's try to apply some of this to this week's Tidy Tuesday

This week's tidy tuesday data set is from NASA about meteorites

```
meteorites <- readr::read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data
```

```
## Parsed with column specification:
## cols(
##   name = col_character(),
##   id = col_double(),
##   name_type = col_character(),
##   class = col_character(),
##   mass = col_double(),
##   fall = col_character(),
##   year = col_double(),
##   lat = col_double(),
##   long = col_double(),
##   geolocation = col_character()
## )
```

Now let's take a look at a summary of the dataframe

```
summary(meteorites)
```

```
##      name                id          name_type            class
##  Length:45716       Min.   :    1   Length:45716       Length:45716
##  Class :character   1st Qu.:12689   Class :character   Class :character
##  Mode  :character   Median :24262   Mode  :character   Mode  :character
##                     Mean   :26890
##                     3rd Qu.:40657
##                     Max.   :57458
##
##       mass              fall                year           lat
##  Min.   :       0   Length:45716       Min.   : 860   Min.   :-87.37
```

```
##  1st Qu.:       7   Class :character   1st Qu.:1987   1st Qu.:-76.71
##  Median :      33   Mode  :character   Median :1998   Median :-71.50
##  Mean   :   13278                      Mean   :1992   Mean   :-39.12
##  3rd Qu.:     203                      3rd Qu.:2003   3rd Qu.:  0.00
##  Max.   :60000000                      Max.   :2101   Max.   : 81.17
##  NA's   :131                           NA's   :291    NA's   :7315
##       long         geolocation
##  Min.   :-165.43   Length:45716
##  1st Qu.:   0.00   Class :character
##  Median :  35.67   Mode  :character
##  Mean   :  61.07
##  3rd Qu.: 157.17
##  Max.   : 354.47
##  NA's   :7315
```

Looks like we have some bad data, the max year is 2101, I find this suspicious. I am going to filter out anything greater than the current year.

```
meteorites <- filter(meteorites, year <= 2019)
```

Let's see how many meteorites have been found vs. fell each year

```
meteorite_counts <- meteorites %>% group_by(fall, year) %>% summarise(count= n())
print(meteorite_counts)
```

```
## # A tibble: 450 x 3
## # Groups:   fall [2]
##    fall   year count
##    <chr> <dbl> <int>
##  1 Fell    860     1
##  2 Fell    920     1
##  3 Fell   1399     1
##  4 Fell   1490     1
##  5 Fell   1491     1
##  6 Fell   1495     1
##  7 Fell   1519     1
##  8 Fell   1583     1
##  9 Fell   1621     1
## 10 Fell   1623     1
## # ... with 440 more rows
```

So now we're considering a year an observation and we want to adjust our dataframe so that each year is on one row only. We use the *spread()* function from *tidyr*

```
spread_meteorite<-meteorite_counts %>% spread(key=fall, value=count)
```

There are a number of years where we have a meteor that fell but none that were found, spread automatically fills this with an NA but we want to just fill it with zero because given our dataframe we will assume if there is no record there is 0 found/fallen that year. We can do this by setting the fill argument to 0 inside spread.

```
spread_meteorite<-meteorite_counts %>% spread(key=fall, value=count, fill=0)
```

What proportion of the meteorites in each year were 'Found' as opposed to 'Fell', lets calculate this in a new column.

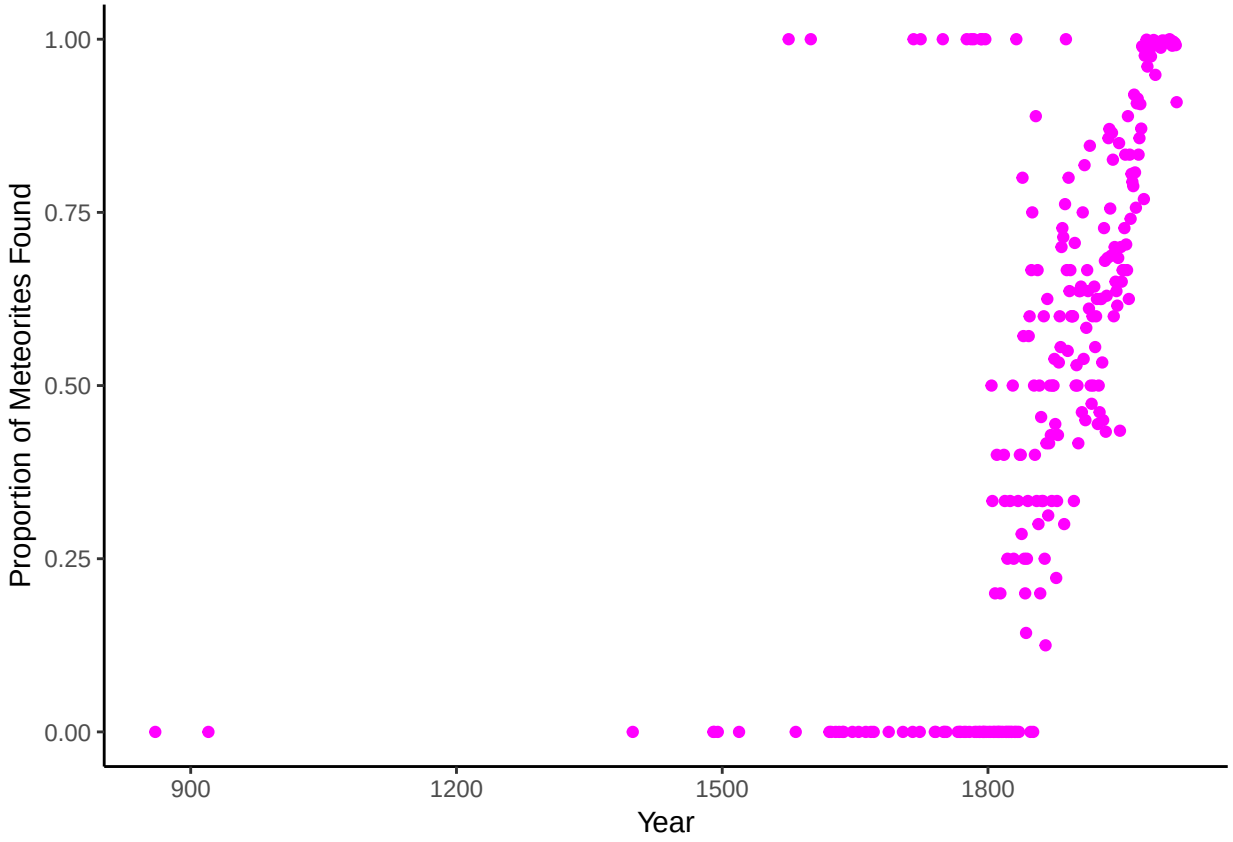
```
spread_meteorite<-spread_meteorite %>% mutate(prop_found = Found/(Fell + Found))
spread_meteorite
```

```
## # A tibble: 264 x 4
```

```
##      year  Fell Found prop_found
##     <dbl> <dbl> <dbl>      <dbl>
##  1   860      1     0          0
##  2   920      1     0          0
##  3  1399      1     0          0
##  4  1490      1     0          0
##  5  1491      1     0          0
##  6  1495      1     0          0
##  7  1519      1     0          0
##  8  1575      0     1          1
##  9  1583      1     0          0
## 10  1600      0     1          1
## # ... with 254 more rows
```

Now lets plot the proportion of found meteors over time.

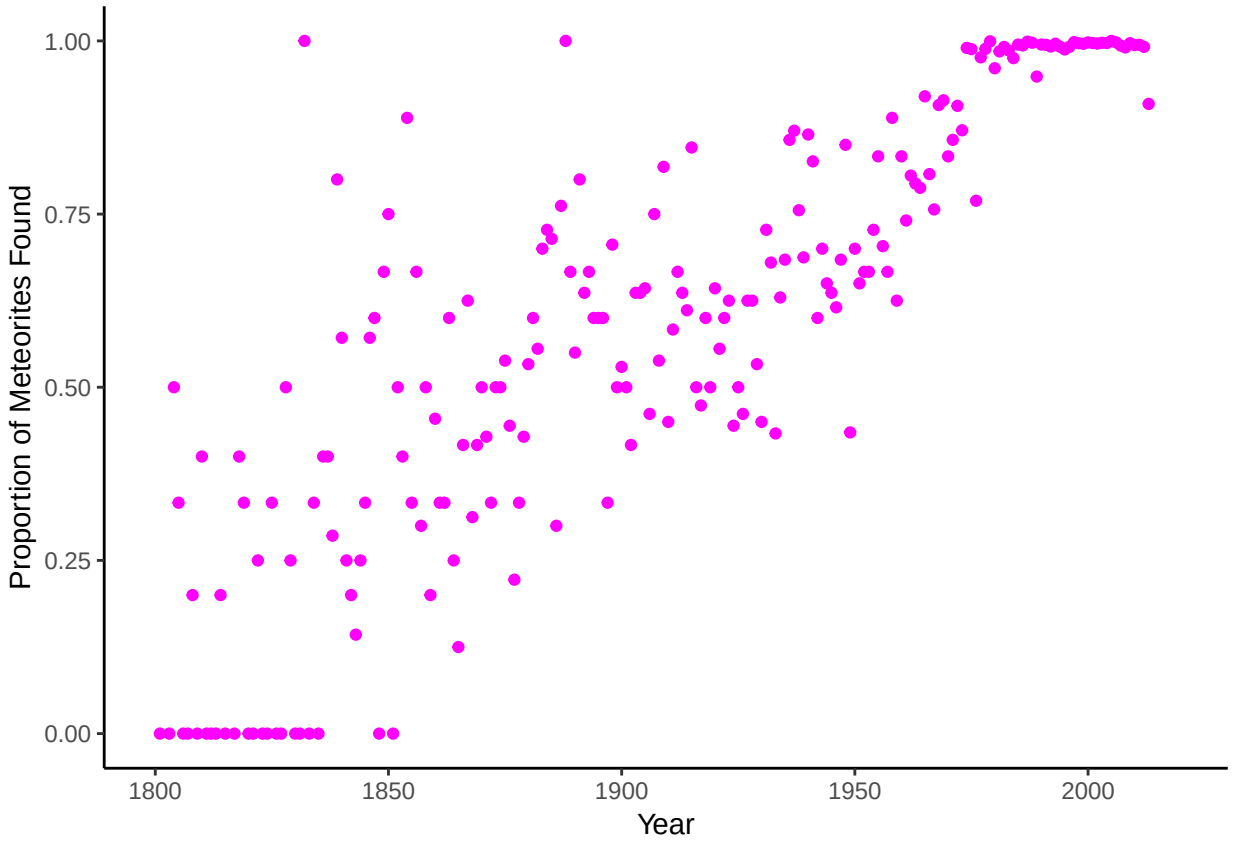
```
ggplot(data=spread_meteorite)+
  geom_point(aes(x=year, y=prop_found), color = 'magenta')+
  theme_classic()+
  xlab('Year')+
  ylab('Proportion of Meteorites Found')
```



It looks like things get more interesting after 1800, let's limit the x axis to be between 1800 and 2019.

```
ggplot(data=spread_meteorite)+geom_point(aes(x=year, y=prop_found), color = 'magenta')+theme_classic()+
```

```
## Warning: Removed 53 rows containing missing values (geom_point).
```



Some ideas for your Tidy Tuesday!

- Do the classes seem to vary by mass? (maybe try a Bar graph or boxplot)
- Is the proportion found vs. fallen different across the classes?
- Create a bar graph with the number of meteorites in each class (check out function top_n() to limit classes)
- Create a graph of median and/or mean meteorite masses over the years
- Create a histogram of the meteor masses