Cleaning Data in tidyr and Overplotting in ggplot2

Emeka Nwosu

September 7, 2016

Introduction and Setup

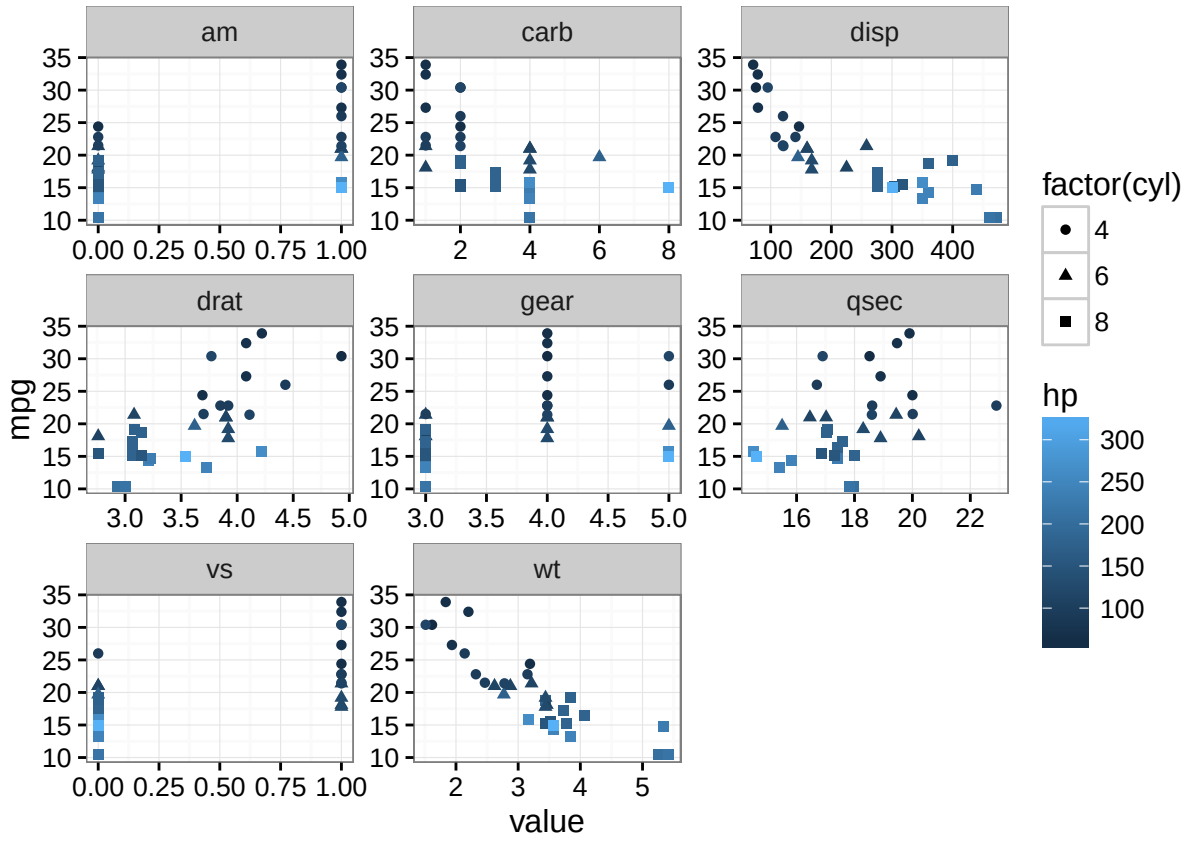
I found an article on r-bloggers about plotting multiple variables using tidyr's "gather" function. Below is the code and the plot shown in the article.

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.3.1
```

```
library(ggplot2)
library(tidyr)

mtcars %>%
  gather(-mpg, -hp, -cyl, key = "var", value = "value") %>%
  ggplot(aes(x = value, y = mpg, color = hp, shape = factor(cyl))) +
    geom_point() +
    facet_wrap(~ var, scales = "free") +
    theme_bw()
```



I impatiently spent about an hour trying to understand every detail of how that code produced that plot

when I could have simply continued reading another paragraph or two to find the exact explanation that I came up with on my own. Anyway, I was very impressed with this technique and wanted to not just copy the code and see it happen in my own R session, but apply it to my own data, mess around with some modifications, and really feel it out.

I've had some data from the Pokemon api for a while now. I haven't come up with any ideas of what to do with it, but I figured with so many potential variables, I could use this as an opportunity to mess around with it and hopefully inspire myself to get to work on something bigger.

The Pokemon api data I used was forked from Github and downloaded to my computer in that folder is a folder called "data" and another called "csv" which holds all the csv files I needed. The data are organized in a very clean manner. Many tables reference others tables. For example the table for pokemon base stats has stat categories coded by numbers. Those numbers are referenced in another table called "stat_names." That table has the coded numbers and names for all the types of stats for each pokemon. Then each of those names (hp, defense, attack, etc) are listed in each of the languages the Pokemon games are translated in. So there's another table called "languages" that has the list of languages and their coded numerical values. I don't have too much experience using SQL, but it reminded me of its foreign keys; tables referencing other tables.

What I didn't expect was how difficult it was setting up the data in such a way that worked the way I wanted it to. This is far from "big data" and I know the hurdles I overcame are nothing for the many people out there with more skill and experience, but it was my first encounter and I had to put in some work.

Process

Loading Data

I needed to piece together parts of data from different tables and combine them into one. I figured there may be some workaround making a plot with data from multiple tables, but I wasn't feeling adventurous.

```
## loading files
## pokemon
pokemon_species <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/pokemon
## pokemon stats
pokemon_stats <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/pokemon_s
## pokemon stat names
stat_names <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/stat_names.cs
## pokemon types
pokemon_types <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/pokemon_ty
## pokemon type names
type_names <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/type_names.cs
## languages
languages <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/languages.csv
## generations
generations <- read.csv("~/Emekas Documents/R-Directory/Don Data/pokeapi-master/data/v2/csv/generations
```

I wanted to get the bulk of the data copied in a dataframe separate from the files in the api folder.

```
df <- pokemon_species[,c("id", "identifier", "generation_id", "growth_rate_id")]
```

Adding Pokemon Stats

Next I started working with the languages data. English is coded as "9" so I filtered out all the stat names in different languages. Then I used a function to replace the coded stat names (1, 2, 3.) with their written word equivalents (hp, attack, defense..). Then I filtered the pokemon down to the 721 "actual" pokemon (some

extra ones like Misingno I think are coded) and only the columns I was interested in. Finally I did a simply column bind add that data to the existing dataframe.

```
## subsetting stat names
## 9 represents english in the languages dataframe
stat_names2 <- dplyr::filter(stat_names, local_language_id == 9)

## replacing data ish
for (i in 1:length(stat_names2$stat_id)){
  pokemon_stats$stat_id <- gsub(pattern = i, replacement = stat_names2$name[i], x = pokemon_stats$stat_
}

## cleaning pokemon stats df
pokemon_stats2 <- dplyr::filter(pokemon_stats, pokemon_id <= 721)
pokemon_stats2 <- spread(subset(pokemon_stats2, select = -c(effort)), stat_id, base_stat)

df <- cbind(df, pokemon_stats2[,2:7])
```

Adding Pokemon Types

Next I had to get the pokemon types (fire, water, dragon, etc for those who don't know) into the data frame. Some pokemon have two types, and for those, I just took the type in slot 1. I was more concerned with getting this plot to work than making a perfect Pokemon plot. The code is mostly the same as with the stats section above.

The main hiccup I came across was with the use of gsub. This may get complicated, but I'll do my best to explain it clearly. The stats and types are coded as 1, 2, 3, etc. The written word type names are in the new "type_names2"" dataframe I made listing the written word type names (few, water, rock, etc.) with their corresponding number (1, 2, 3, etc.). So with a for loop, I replaced each number in the pokemon_type dataframe with its written word equivalent. So it starts with 1, and replaces every "1" found in pokemon_type with its corresponding type name. Then does it with 2 and 3 and so on.

The problem here is that the number "12" would also get partially replaced. If 1 = "normal" and 2 = "fighting" then 12 = normalfighting and not "grass" (or whatever 12 actually corresponds to). It would replace the "1" then in the second loop, it would replace the "2." Then by the time it got to "12" they were all gone. To fix this, I reversed the for loop to start at the highest number and work backwards to 1. This way the 16's and 15's and such would get substituted with their written word equivalents first and it wouldn't affect the single digit numbers. It may seem a little weird to wrap your head around it if you aren't comfortable with coding, but it does make sense and it did work.

```
## subsetting type names
## 9 represents english in the languages dataframe
type_names2 <- dplyr::filter(type_names, local_language_id == 9)

## cleaning pokemon stats df
pokemon_types2 <- dplyr::filter(pokemon_types, pokemon_id <= 721)
pokemon_types2 <- dplyr::filter(pokemon_types2, slot == 1)

## replacing data ish
for (i in length(type_names2$type_id):1){
  pokemon_types2$type_id <- gsub(pattern = type_names2$type_id[i], replacement = type_names2$name[i], x
}

df <- cbind(df, pokemon_types2[2])
```

The Message

Setting all that up took me so long that I actually forgot what I was trying to plot in the first place. Writing all that took me so long I completely forgot to explain what I wanted to plot with the data so I'll do that now.

The code for the example plot from R-Bloggers works like this. I'm going to explain assuming you already know how the "gather" function works. The mtcars dataset has the variables mpg, hp, and cyl listed with "-" in front as well as the titles of all those plots "am," "carb," etc. gather turns wide data into long data. Normally, "gather" would turn all data from wide to long, but adding the "-" sign in front of a variable tells it to exclude it from the transformation. Basically the excluded variables are the independent Y variables and the rest are dependent X variables. So you can compare any or all of the variables not listed with a "-" to any or all of the variables listed with a "-".

So I had to decide which variables I wanted for independent variables and which ones I wanted for dependent variables. I decided to use growth_rate as my primary independent variable because it was the only that made sense. In an actual real-world application, it's entirely not interesting and was only used that way because it works in a way that makes sense statistically. "Generation" was a good factor variable to use and "type" is perfect for a categorical variable. Then the dependednt variables which youll see titled in the plots are the pokemon's stats, Attack, Defense, HP, Special Attack, Special Defense, and Speed I want to see how each of the pokemon's stat categories relate to the pokemon's growth rate while accounting for the generation and type of the pokemon.

Actually looking at the plot again now, I should have associated "generation" with coloring because there is actually an order to it where as "type" should have been done in shapes, but whatever. Too late now. You'll see what I mean at the end.

```
summary(df)
```

```
##        id           identifier  generation_id   growth_rate_id
##  Min.   :  1   abomasnow :  1   Min.   :1.000   Min.   :1.000
##  1st Qu.:181   abra      :  1   1st Qu.:2.000   1st Qu.:2.000
##  Median :361   absol     :  1   Median :3.000   Median :2.000
##  Mean   :361   accelgor  :  1   Mean   :3.323   Mean   :2.563
##  3rd Qu.:541   aegislash :  1   3rd Qu.:5.000   3rd Qu.:4.000
##  Max.   :721   aerodactyl:  1   Max.   :6.000   Max.   :6.000
##                (Other)   :715
##      Attack          Defense             HP         Special Attack
##  Min.   :  5.00   Min.   :  5.00   Min.   :  1.00   Min.   : 10.00
##  1st Qu.: 53.00   1st Qu.: 50.00   1st Qu.: 50.00   1st Qu.: 45.00
##  Median : 74.00   Median : 65.00   Median : 65.00   Median : 65.00
##  Mean   : 74.99   Mean   : 70.84   Mean   : 68.38   Mean   : 68.71
##  3rd Qu.: 95.00   3rd Qu.: 85.00   3rd Qu.: 80.00   3rd Qu.: 90.00
##  Max.   :165.00   Max.   :230.00   Max.   :255.00   Max.   :154.00
##
##  Special Defense      Speed          type_id
##  Min.   : 20.00   Min.   :  5.00   Length:721
##  1st Qu.: 50.00   1st Qu.: 45.00   Class :character
##  Median : 65.00   Median : 65.00   Mode  :character
##  Mean   : 69.32   Mean   : 65.71
##  3rd Qu.: 85.00   3rd Qu.: 85.00
##  Max.   :230.00   Max.   :160.00
##
```

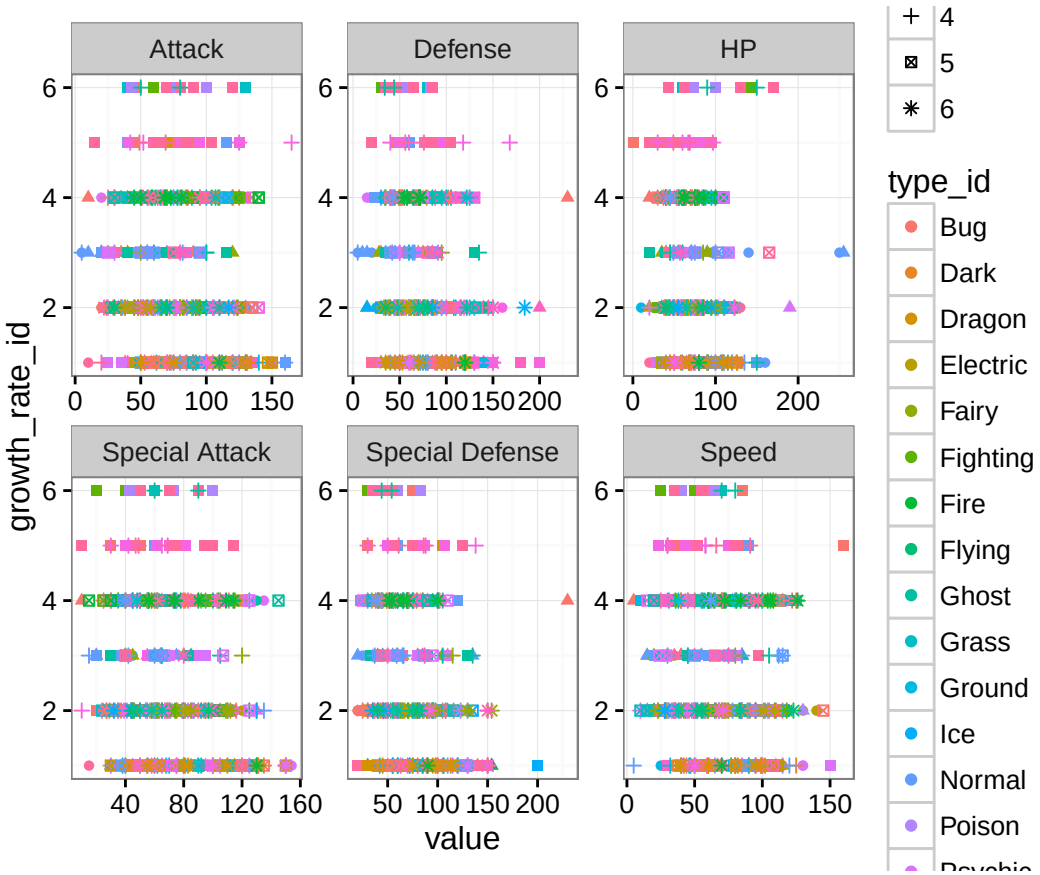
```
head(df)
```

```
##   id identifier generation_id growth_rate_id Attack Defense HP
## 1  1  bulbasaur             1              4     49      49 45
## 2  2    ivysaur             1              4     62      63 60
## 3  3   venusaur             1              4     82      83 80
## 4  4 charmander             1              4     52      43 39
## 5  5 charmeleon             1              4     64      58 58
## 6  6  charizard             1              4     84      78 78
##   Special Attack Special Defense Speed type_id
## 1             65              65    45   Grass
## 2             80              80    60   Grass
## 3            100             100    80   Grass
## 4             60              50    65    Fire
## 5             80              65    80    Fire
## 6            109              85   100    Fire
```

The Plot

Finally, the moment you've all been waiting for.

```
df %>%
  gather(-id, -identifier, -generation_id, -growth_rate_id, -type_id, key = "stat", value = "value") %>%
  ggplot(aes(x = value, y = growth_rate_id, color = type_id, shape = factor(generation_id))) +
    geom_point() +
    facet_wrap(~ stat, scales = "free") +
    theme_bw()
```



So. . . .yeah. The plot ended up a giant mess. In the words of Kramer, "It's too busy." There are just way too many categories and variables and data points all happening at once. It's possible to make sense of it all, but it really needs to be simplified in order to be readable.

Conclusion

This was all more of a learning experience than an "I'm going to add something cool to my portfolio of work." Digging through and setting up the data took 99% of my time. It wasn't fun, but it's great that I experience that because every data scientist will tell you that a vast majority of their time is spent cleaning data rather than the analysis, the reason why people want to get into data science. The second thing I learned was overplotting. I'm still not sure if that's an actual word, but I'll keep using it. I wanted to make the coolest plot by throwing everything I could at it, but I put too much into it and the message got lost. I'm not even sure what that message was anymore. It was still a worthwhile experience despite not ending up with useful.