Broom: The R Package That Tidies Your Model Outputs

Katherine Li

November 26, 2017

At the beginning of the semester, we were assigned to read Hadley Wickham's paper on tidy data, which talks about the importance of tidying data for further data analysis. Wickham laments that although there exist tools for tidying raw data – including tools within the tidyverse such as readr, dplyr, and ggplot2 that we've studied quite carefully in this class – there are few tools that tidy the outputs of models we apply to our newly tidied raw datasets. I found this point interesting; much like tidying raw data, tidying model outputs is not something I (and I believe many others too) actively consider as I am performing statistical analyses in R. I decided to make this second post about broom, an R package that serves this purpose, because I believe this to be a helpful step in the data analyses cycle and because I am fascinated by the features of this package. In the rest of my post, I discuss in greater detail what broom is and its functions; I then talk about the packages with which it is commonly used and conclude with some examples on how broom is used. My chosen dataset for conveying these points is mtcars, as it is built in to R and easily accessible to anyone.

What is broom?

Broom, as I mentioned before, turns the outputs of statistical models and shapes them into data frames convenient for input-tidy data manipulation tools. The three functions in broom, as introduced by broom creator David Robinson in his blog on broom, are tidy(), augment(), and glance(); these functions, used in conjunction with the dplyr and ggplot2 packages, provide a rich mix of methods with which to perform data manipulation and analysis. Later in this post, I will offer more details on these specific functions and methods, but for now, I want to briefly mention some of the functions you should expect to see.

How do I use broom?

To start exploring broom, I installed it and loaded it into my session. Because I will be using dplyr and ggplot2 as well, I also loaded those packages:

```
# install package
# install.packages("broom")

# load packages
library(broom)
```

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

```
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(ggplot2)
```

1. Tidy()

I will illustrate use cases of the broom functions through a series of short examples. I arrange examples that more thoroughly present the broom package – and its interactions with dplyr and ggplot2 – at the end, as I find it helpful to understand the individual parts before looking at the whole process. My first example makes use of the broom function tidy(). I apply a linear regression method (although you can apply any modelling method) to my dataset mtcars and use tidy() to transform it into a tidier format.

I'd like to emphasize again that my method was arbitrarily chosen, as well as the variables I include in my linear regression. The functions of the broom package can be applied to any modelling output to tidy up the data. My linear regression, which I execute through the linear models function lm(), happens to regress linearly mpg, which stands for miles per gallon, on cyl, the number of cylinders; hp, the gross horsepower; and drat, the rear axle ratio. This regression looks at whether variations in mpg can be well-explained by variations in the latter three variables. I include summary statistics of this regression to demonstrate how the data looks ***before*** its transformation: Does it look easy to manipulate?

```
# linear regression method on data
linear <- lm(mpg ~ cyl + hp + drat, data = mtcars)

# summary statistics from my linear regression method
summary(linear)
```

```
##
## Call:
## lm(formula = mpg ~ cyl + hp + drat, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2103 -2.0384 -0.0944  1.2891  6.7107
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.51406    7.99354   2.817   0.0088 **
## cyl         -1.36060    0.73493  -1.851   0.0747 .
## hp          -0.02878    0.01530  -1.881   0.0704 .
## drat         2.84090    1.52208   1.866   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.045 on 28 degrees of freedom
## Multiple R-squared:  0.7694, Adjusted R-squared:  0.7447
## F-statistic: 31.14 on 3 and 28 DF,  p-value: 4.616e-09
```

Clearly, the summary statistics are all over the place and they can be better formatted. One possible way of formatting these statistics is to create separate columns for different statistics – p-values, standard errors, and other metrics. Let's examine how tidy() organizes our output:

```
# apply tidy to linear regression statistics
tidy(linear)
```

```
##          term    estimate  std.error statistic     p.value
## 1 (Intercept) 22.51405645 7.99353801  2.816532 0.008797271
## 2         cyl -1.36059873 0.73492694 -1.851339 0.074695109
## 3          hp -0.02878368 0.01529927 -1.881376 0.070356092
## 4        drat  2.84090154 1.52207530  1.866466 0.072481940
```

The chart it produces cleanly presents the data and satisfies all of Hadley Wickham's requirements for "tidy data". It is in analytical form: all the major output categories are displayed neatly in each column: the estimated coefficient for each of my three predictor variables and an intercept term, the variation of these coefficients, and their significance in explaining the variation of my response variable mpg as measured by the respective p-values. This contrasts starkly with my previous display of these statistics, which has complicated metric names and is generally incompatible with many tidyverse tools. In short, tidy() enables you to convert messy data into a manipulation-friendly format.

2. Augment()
The function augment() allows you to add your newly calculated statistics seamlessly back into your original dataset of only the variables I used in my linear regression model and the rownames, which are the various cars. To illustrate, I display the first few rows of my mtcars dataset with fitted statistics:

```
# display data frame of calculated statistics along with original data
head(augment(linear))
```

```
##           .rownames  mpg cyl  hp drat  .fitted   .se.fit    .resid
## 1         Mazda RX4 21.0   6 110 3.90 22.26377 0.8961358 -1.263775
## 2     Mazda RX4 Wag 21.0   6 110 3.90 22.26377 0.8961358 -1.263775
## 3        Datsun 710 22.8   4  93 3.85 25.33225 0.9746398 -2.532250
## 4    Hornet 4 Drive 21.4   6 110 3.08 19.93424 0.9800888  1.465765
## 5 Hornet Sportabout 18.7   8 175 3.15 15.54096 0.8845073  3.159038
## 6           Valiant 18.1   6 105 2.76 19.16907 1.3696238 -1.069065
##         .hat   .sigma     .cooksd .std.resid
## 1 0.08659411 3.090720 0.004468696 -0.4342183
## 2 0.08659411 3.090720 0.004468696 -0.4342183
## 3 0.10243042 3.058221 0.021977916 -0.8776934
## 4 0.10357896 3.086834 0.007465460  0.5083684
## 5 0.08436136 3.035398 0.027069852  1.0840847
## 6 0.20227551 3.092612 0.009793247 -0.3930503
```

The resultant data frame highlights the specific features of each car in relation to the regression model. A comparison between the columns of my new data frame and the previous two displays of output data suggests that the metrics displayed are not the same. Augment() not only shapes the data into a good analytical form but also tailors the values that are displayed to the variables I choose for my data frame. For example, the fitted values and their standard deviations are displayed for each car because it makes for an easy side-by-side comparison of my true mpg value with the estimated value. This paves the way for further analysis on the differences between my true and estimated values.

3. Glance()
The glance() function produces a data frame of useful data statistics in the data analysis process. For example, it reports the R^2 number and p-value, which are typically cited in reports, and the log likelihood statistic, which is important for intermediate calculations. This is the data frame produced from applying glance() to my linear regression outputs:

```
# outputs a data frame of commonly cited statistics
glance(linear)
```

```
##   r.squared adj.r.squared    sigma statistic      p.value df    logLik
## 1 0.7693992      0.744692 3.045297  31.14066 4.616221e-09  4 -78.90468
##        AIC     BIC deviance df.residual
## 1 167.8094 175.138 259.6673          28
```

Glance() offers a facet of statistical analysis the other two functions do not offer: intermediate statistics and considerations that are made while performing a linear regression. Combined, these three functions provide a comprehensive view of my regression model.

Why use broom with dplyr and ggplot2?

Broom is an intermediate step for the kinds of manipulations we aim to make through dplyr and ggplot2. In class, we practiced transforming raw data into a data frame alongside using dplyr and ggplot2 tools. Likewise, broom, dplyr and ggplot2 make up a process, characterized by conversion into a data frame through broom, shaping the data frame through dplyr, producing a graph through ggplot2. Broom is designed to work well with these two packages.

This is a useful manual on broom and dplyr. It talks about how best to use broom and dplyr to gain insight into your data. Many cases it introduces involves use of dplyr's group_by() and summarize() functions, methods we have had a lot of practice with in class. The manual also provides examples of how to input the data frame into ggplot(), something we have also had much experience with through class.

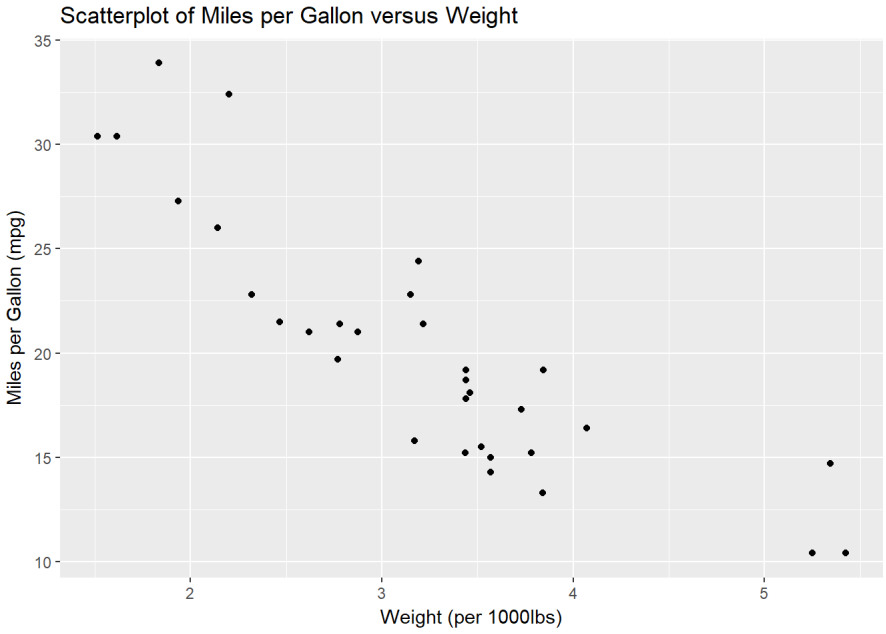
Can I see some examples of broom in action?

I've researched countless examples of broom being used in more complex analyses than the basic linear regression examples I discussed earlier in my post. Consequently, I've compiled two examples I find most interesting to showcase the value of broom as a statistical tool.

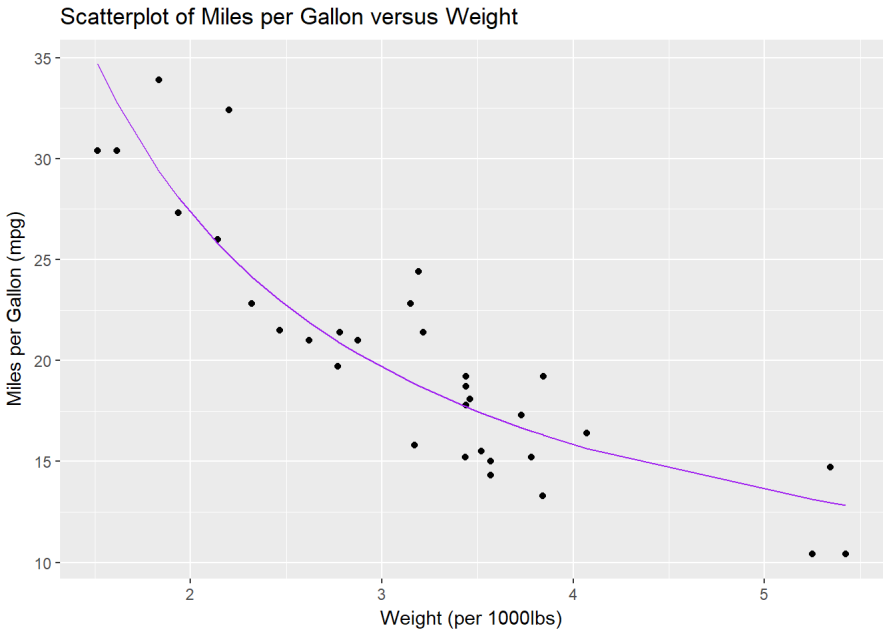
Example 1

My first example is a bootstrapping example. I arbitrarily chose two variables wt and mpg from my mtcars dataset to graphically analyze. My first step was to create a regular scatterplot of mpg versus wt – I wanted to get an initial idea of what my datapoints looked like and what model I imagine would best fit the data. The graph shows a slightly nonlinear relationship between wt and mpg; I fitted a nonlinear line to the graph to see how well it matches the data and it seems to match my datapoints reasonably well.

```
# get some initial ideas of the two parameters wt and mpg graphically
graph0 <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point() + ggtitle("Scatterplot of Miles per Gallon versus We
ight") + xlab("Weight (per 1000lbs)") + ylab("Miles per Gallon (mpg)")
graph0
```




```
# fit line to initial scatterplot
fitline <- nls(mpg ~ k / wt + b, mtcars, start = list(k = 1, b = 0))
graph0 + geom_line(aes(y = predict(fitline)), color = "purple")
```



Before beginning the bootstrap procedure, it is helpful to quickly look over the summary statistics of this fitted, nonlinear model. A brief scan can help us get more familiar with the data or determine whether there are changes we might want to make to our line. There is nothing that appears especially out of the ordinary – in fact, the p-values displayed confirm that our constants are not zero – so we proceed to use broom manipulations to bootstrap.

```
# get summary statistics of fitted line
summary(fitline)
```

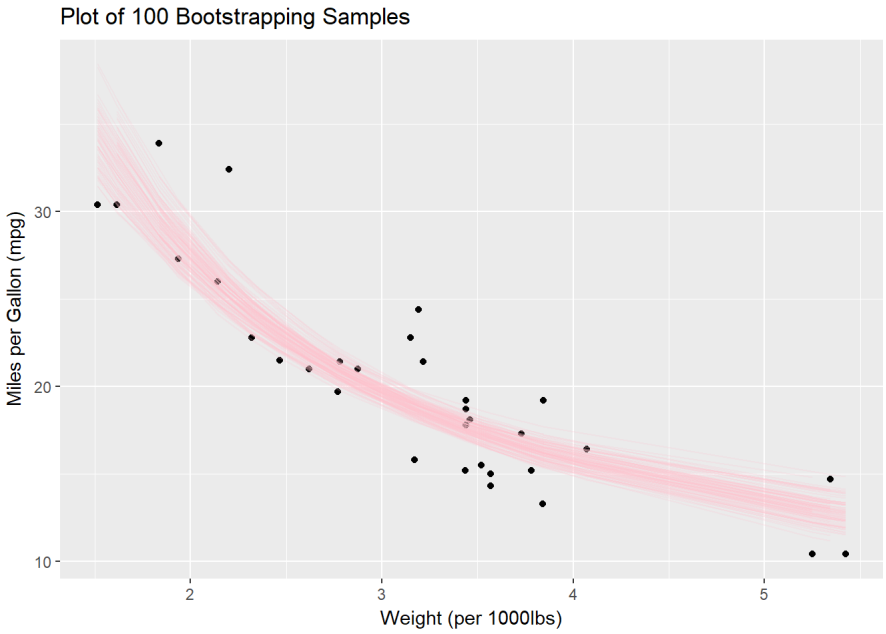
```
##
## Formula: mpg ~ k/wt + b
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## k   45.829      4.249  10.786 7.64e-12 ***
## b    4.386      1.536   2.855  0.00774 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.774 on 30 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 2.877e-08
```

In order to maintain reproducibility in my bootstrap sampling procedure, I set my identifier as 123 in the set.seed() function. I then begin the bootstrap, using the dplyr pipeline to produce 100 bootstrap samples (with replacement) from my mtcars data frame. The do() function in the dplyr package enables application of each of these 100 samples to the nls() function I applied earlier to calculate my nonlinear test graph. The dot arguments in the nls() graph represent areas where I want to insert each bootstrapped sample as my dataset. Augment() then turns the output of each of these 100 models into a data frame for future manipulation.

```
# set seed for bootstrapping
set.seed(123)

# function creates augmented data frames for 100 random samples
bootaug <- mtcars %>% bootstrap(100) %>% do(augment(nls(mpg ~ k / wt + b, ., start = list(k = 1, b = 0)), .))

# visualize these bootstrapped data frames through ggplot
ggplot(bootaug, aes(x = wt, y = mpg)) + geom_point() + geom_line(aes(y = .fitted, group = replicate), alpha = 0.2,
col = "pink") + ggtitle("Plot of 100 Bootstrapping Samples") + xlab("Weight (per 1000lbs)") + ylab("Miles per Gall
on (mpg)")
```



After storing the information from my 100 bootstrap sample models in a data frame through my augment() function, I decided to produce a visualization to illustrate how broom and ggplot2 work together in data analysis. The ggplot() scatterplot function is implemented the same way we studied in class – I used geom_point() to create the scatterplot and geom_line() to fit a nonlinear graph – so I will not elaborate more on the ggplot() part.
I will however briefly point out how tight the graphs produced from the 100 bootstraps ended up being, suggesting greater certainty for the fitted nonlinear graph.

Example 2

The second example I thought would be cool to examine for this post has to do with k-means clustering, in which a dataset is clustered into k different groups with k separate means. I use the kmeans() function on the mtcars dataset, also specifying that I want my dataset clustered into two groups as an argument for this function. The results are as follows:

```
# get k-means clusters from data
kcluster <- kmeans(mtcars, 2)
kcluster
```

```
## K-means clustering with 2 clusters of sizes 18, 14
##
## Cluster means:
##        mpg      cyl     disp        hp     drat       wt     qsec
## 1 23.97222 4.777778 135.5389  98.05556 3.882222 2.609056 18.68611
## 2 15.10000 8.000000 353.1000 209.21429 3.229286 3.999214 16.77214
##          vs        am     gear     carb
## 1 0.7777778 0.6111111 4.000000 2.277778
## 2 0.0000000 0.1428571 3.285714 3.500000
##
## Clustering vector:
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##                   1                   1                   1
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##                   1                   2                   1
##          Duster 360           Merc 240D            Merc 230
##                   2                   1                   1
##           Merc 280           Merc 280C          Merc 450SE
##                   1                   1                   2
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##                   2                   2                   2
## Lincoln Continental   Chrysler Imperial            Fiat 128
##                   2                   2                   1
##         Honda Civic      Toyota Corolla       Toyota Corona
##                   1                   1                   1
##    Dodge Challenger         AMC Javelin          Camaro Z28
##                   2                   2                   2
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2
##                   2                   1                   1
##        Lotus Europa      Ford Pantera L        Ferrari Dino
##                   1                   2                   1
##       Maserati Bora          Volvo 142E
##                   2                   1
##
## Within cluster sum of squares by cluster:
## [1] 58920.54 93643.90
##  (between_SS / total_SS =  75.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

From this model output, it appears the kmeans() function clustered my mtcars dataset by dividing the cars into two separate groups and calculating the statistics within each cluster. I can get an even better idea of the characteristics of these groups by calling the summary() function.

```
# output the summary statistics of the kcluster
summary(kcluster)
```

```
##              Length Class  Mode
## cluster      32     -none- numeric
## centers      22     -none- numeric
## totss         1     -none- numeric
## withinss      2     -none- numeric
## tot.withinss  1     -none- numeric
## betweenss     1     -none- numeric
## size          2     -none- numeric
## iter          1     -none- numeric
## ifault        1     -none- numeric
```

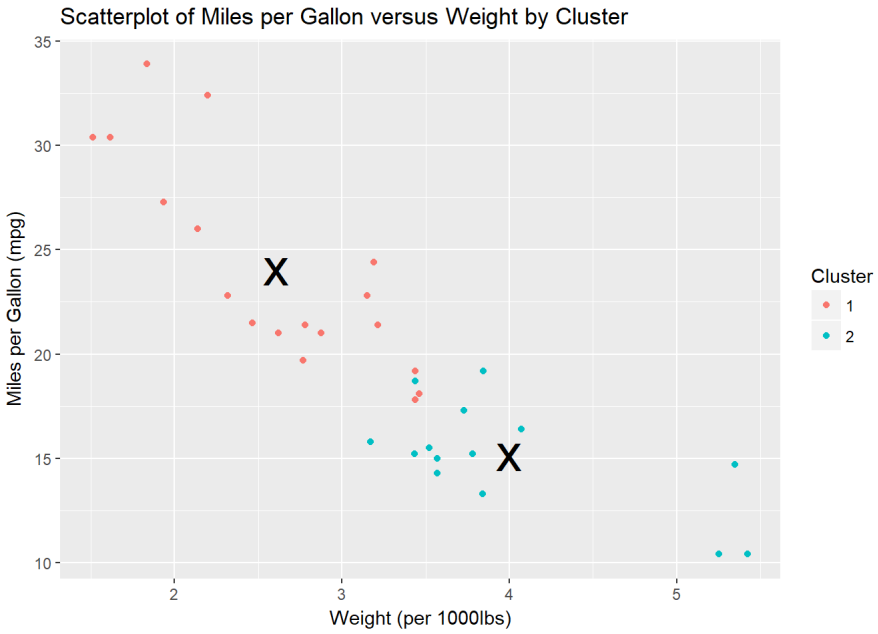
This summary table suggests three different types of measurements on my dataset. The cluster and centers components apply to individual pieces: there are 32 different cars that can be split into the 2 clusters and there are 22 different centers of car characteristics such as mpg or cyl given for both clusters. The withinss and size components apply to each cluster; and the totss, tot.withinss, betweenss, iter, and ifault metrics apply to all the data as a whole. It is easier to understand my future graphical notations when keeping these features in mind.

Now that I have my model output kcluster of my kmeans() function, I can use broom to prepare it for further analysis. I first use tidy to convert some basic metrics into a data frame, which I call tidycluster. I then resort to visualization again through ggplot(), with which I can show how the datapoints cluster and mark where they cluster within my framework of plotting mpg against wt. The dataset I insert into ggplot() makes use of the augment() function because in order to show the relationship between wt and mpg from my original data along with my newly formed clusters, I need both sets of information to be placed within one data frame. This is exactly what augment() does. I use the symbol “X” to represent the location of each cluster, by calling geom_point() again but specifying that I would like it to take the tidycluster dataset, which specifies two cluster categories, and have these two cluster points be marked as “X”’s. The rest of my ggplot() code involves the same manipulations we’ve performed in class: using ggtitle() to set a title, xlab() and ylab() to specify axis names, geom_point() on my augmented data to create the scatterplot, and scale_color_discrete() to change the legend title. Because of this, I do not feel the need to go into detail about these specifics in this post.

```
# tidy the kcluster output summary
tidycluster <- tidy(kcluster)
colnames(tidycluster) <- c("mpg", "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am", "gear","carb", "cluster s
ize", "sum of squares", "cluster")
tidycluster
```

```
##        mpg      cyl     disp        hp     drat       wt     qsec
## 1 23.97222 4.777778 135.5389  98.05556 3.882222 2.609056 18.68611
## 2 15.10000 8.000000 353.1000 209.21429 3.229286 3.999214 16.77214
##          vs        am     gear     carb cluster size sum of squares
## 1 0.7777778 0.6111111 4.000000 2.277778           18       58920.54
## 2 0.0000000 0.1428571 3.285714 3.500000           14       93643.90
##   cluster
## 1       1
## 2       2
```

```
# visualization of cluster statistics as organized using tidy
ggplot(augment(kcluster, mtcars), aes(x = wt, y = mpg)) + geom_point(data = tidycluster, size = 10, shape= "x") +
geom_point(aes(color = .cluster)) + ggtitle("Scatterplot of Miles per Gallon versus Weight by Cluster") + xlab("We
ight (per 1000lbs)") + ylab("Miles per Gallon (mpg)") + scale_color_discrete("Cluster")
```



I next want to show how broom functions can be helpful for understanding how results can vary as the number of clusters for a given dataset varies. I make use of the do() function, which I described above, to accomplish this purpose.

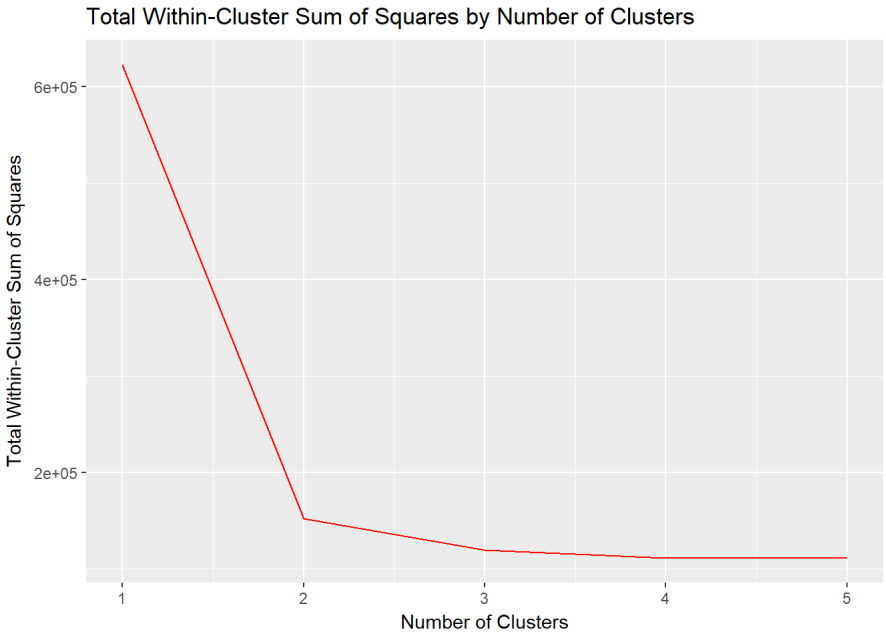
I first create an allclusts data frame that contains output from the kmeans() model for different numbers of clusters, the maximum cluster number, in my case, being 5. For this analysis, I wanted to focus on some of the major statistics used in my model, so I invoked the glance() function on each of my sub-data frames as specified by number of clusters. My graph, which I created through ggplot(), shows how total within cluster sum of squares will vary with the number of clusters created. My plot suggests that there is an inverse relationship between total within cluster sum of squares and number of clusters. The more clusters there are, the lower the overall within cluster sum of squares tends to be.

```
# now apply this k times to k different numbers of clusters
allclusts <- data.frame(k = 1:5) %>% group_by(k) %>% do(allclust = kmeans(mtcars, .$k))

# tidy data using glance
glancecluster <- allclusts %>% group_by(k) %>% do(glance(.$allclust[[1]]))
```

```
## Warning: Grouping rowwise data frame strips rowwise nature
```

```
# graphical representation of glance statistic using ggplot
ggplot(glancecluster, aes(x = k, y = tot.withinss)) + geom_line(color = "red") + ggtitle("Total Within-Cluster Sum
of Squares by Number of Clusters") + xlab("Number of Clusters") + ylab("Total Within-Cluster Sum of Squares")
```



Conclusion

Through this post, I hope I've convinced you that broom is a package well-worth reading more about. At a basic level, it is certainly extremely helpful for cleaning up messy model outputs. As I have shown through my examples, broom makes data manipulation more convenient by organizing model outputs into data frames that can then be used with dplyr, ggplot2, and other packages from the tidyverse to better analyze data. This package is guaranteed to sweep you off your feet!

References

http://varianceexplained.org/r/broom-intro/

https://cran.r-project.org/web/packages/broom/vignettes/broom_and_dplyr.html

https://cran.r-project.org/web/packages/broom/vignettes/bootstrapping.html

https://cran.r-project.org/web/packages/broom/vignettes/kmeans.html

http://vita.had.co.nz/papers/tidy-data.pdf

https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/mtcars.html

https://www.r-bloggers.com/a-tidy-model-pipeline-with-twidlr-and-broom/

https://stackoverflow.com/questions/35272457/what-does-the-dplyr-period-character-reference

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.