Data Transformation with dplyr

A Look at the Average Price of Avocados in New York

By Anisha BharathSingh

When working on any data science project, you will need to transform data into the proper form to work with. This can include creating new variables, renaming variables or reordering observations in a data frame. Overall, we just want to make data easier to work with. This is where the dplyr package comes in. As part of the tidyverse package, which we are already familar with, dplyr allows us to transform our data.

The five main dplyr functions we will be covering are: filter(), arrange(), select(), mutate() and summarize().

First, let's make sure that the tidyverse package is installed and loaded.

```
#install.packages("tidyverse")
library(tidyverse) #load tidyverse package

## ── Attaching packages ──────────────────────────────────────────
────── tidyverse 1.2.1 ──

## ✔ ggplot2 3.1.0     ✔ purrr   0.2.5
## ✔ tibble  1.4.2     ✔ dplyr   0.7.8
## ✔ tidyr   0.8.2     ✔ stringr 1.3.1
## ✔ readr   1.2.1     ✔ forcats 0.3.0

## ── Conflicts ──────────────────────────────────────────────────
── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

Now for the data. We will be working with a data set that includes avocado prices and more from 2015 through 2018. Let's read the avocado excel file and preview the data set with the head() function to see what variables we have to work with.

```
avocado_data <- read_csv("avocado.csv") #assign data to "avocado_data"

## Warning: Missing column names filled in: 'X1' [1]

## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   Date = col_date(format = ""),
##   AveragePrice = col_double(),
##   `Total Volume` = col_double(),
##   `4046` = col_double(),
##   `4225` = col_double(),
##   `4770` = col_double(),
##   `Total Bags` = col_double(),
##   `Small Bags` = col_double(),
##   `Large Bags` = col_double(),
##   `XLarge Bags` = col_double(),
##   type = col_character(),
##   year = col_double(),
##   region = col_character()
## )
```

```
head(avocado_data) #preview data
```

```
## # A tibble: 6 x 14
##      X1 Date       AveragePrice `Total Volume` `4046` `4225` `4770`
##   <dbl> <date>            <dbl>          <dbl>  <dbl>  <dbl>  <dbl>
## 1     0 2015-12-27         1.33         64237.  1037. 5.45e4   48.2
## 2     1 2015-12-20         1.35         54877.   674. 4.46e4   58.3
## 3     2 2015-12-13         0.93        118220.   795. 1.09e5  130.
## 4     3 2015-12-06         1.08         78992.  1132  7.20e4   72.6
## 5     4 2015-11-29         1.28         51040.   941. 4.38e4   75.8
## 6     5 2015-11-22         1.26         55980.  1184. 4.81e4   43.6
## # ... with 7 more variables: `Total Bags` <dbl>, `Small Bags` <dbl>,
## #   `Large Bags` <dbl>, `XLarge Bags` <dbl>, type <chr>, year <dbl>,
## #   region <chr>
```

This data set contains 14 columns including dates, years, average prices and regions. We also have information on how many bags: small, large and Xlarge were sold along with the total volume (or number) of avocados sold. We have specific information on the Product Lookup codes (PLUs) for Hass Avocados and whether or not the avocado was conventional or organic. Lastly, if you take a look at the first column, you will notice that this just numbers each row. We can eliminate this column by getting right into some data transformation.

The select() function allows us to zoom in on a useful set of data. In this case, we will select all variables except the first column, X1, and assign it to a new data frame called “avocado”. We can select all of these variables by using a “:” between the “Date”" and “region”" columns. In general, we could also select a variable directly by its name or by what the variable names start, contain or end with using: starts_with(), contains(), ends_with().

```
avocado <- select(avocado_data, Date:region) #select data from columns date through region
#select(data frame, variable1:variable2)
colnames(avocado) #print variable names
```

```
##  [1] "Date"         "AveragePrice" "Total Volume" "4046"
##  [5] "4225"         "4770"         "Total Bags"   "Small Bags"
##  [9] "Large Bags"   "XLarge Bags"  "type"         "year"
## [13] "region"
```

We can see that this new avocado data frame excludes the “X1”" column.

Part of transforming data also includes having consistent variable names, so let's rename the remaining columns names so that they are all aligned using the colnames() function to assign new names:

```
colnames(avocado) <- c("Date", "Average_Price", "Total_Volume", "4046", "4225", "4770", "Total_Bags"
, "Small_Bags", "Large_Bags", "XLarge_Bags", "Type", "Year", "Region")
colnames(avocado) #print data frame variable names

##  [1] "Date"          "Average_Price" "Total_Volume"  "4046"
##  [5] "4225"          "4770"          "Total_Bags"    "Small_Bags"
##  [9] "Large_Bags"    "XLarge_Bags"   "Type"          "Year"
## [13] "Region"
```

Now that all of our variable names are capitalized and have "_", let's move into our next function: arrange(). This function allows us to arrange our data observations ordered by a variable or column. Let's arrange our data by"Date"", and the "Average_Price"" as a second factor to see how this function works:

```
avocado <- arrange(avocado, Date, Average_Price) #arrange data frame by Date then Average_Price
# arrange(df, VarName)
head(avocado) #preview data frame

## # A tibble: 6 x 13
##   Date       Average_Price Total_Volume `4046` `4225` `4770` Total_Bags
##   <date>             <dbl>        <dbl>  <dbl>  <dbl>  <dbl>      <dbl>
## 1 2015-01-04          0.65     1048062. 7.71e5 1.78e5  6509.     92499.
## 2 2015-01-04          0.71     1062991. 5.06e5 4.36e5  4379.    115838.
## 3 2015-01-04          0.74     1086364. 6.13e5 3.74e5  9817.     89330.
## 4 2015-01-04          0.75      758119. 4.27e5 1.48e5 15268.    168014.
## 5 2015-01-04          0.77     5144267. 2.75e6 1.76e6 73433.    570683.
## 6 2015-01-04          0.8       317861. 1.34e5 1.21e5  4591.     58639.
## # ... with 6 more variables: Small_Bags <dbl>, Large_Bags <dbl>,
## #   XLarge_Bags <dbl>, Type <chr>, Year <dbl>, Region <chr>
```

We can see that our data frame, which was originally ordered by region, is now ordered first by "Date" then by the "Average_Price" for each date. If we wanted to arrange a column in descending order, we could use the desc() function within the arrange() function on the variable we want to arrange.

Let's now take a look at a more specific subset of the data. How about we take a look at the data for avocados in New York year over year? Since the data set does not have data for all of 2018, let's remove these data rows and focus on data from 15'-17'.

We are able to subset variables by specific values using the filter() function. So, let's filter our avocado data frame to create a new data frame that only contains data for New York in the years 15'-17':

```
NY_15_17 <- filter(avocado, Region == "NewYork", Year != "2018") #filter for New York data only that do
es not contain 2018 data
```

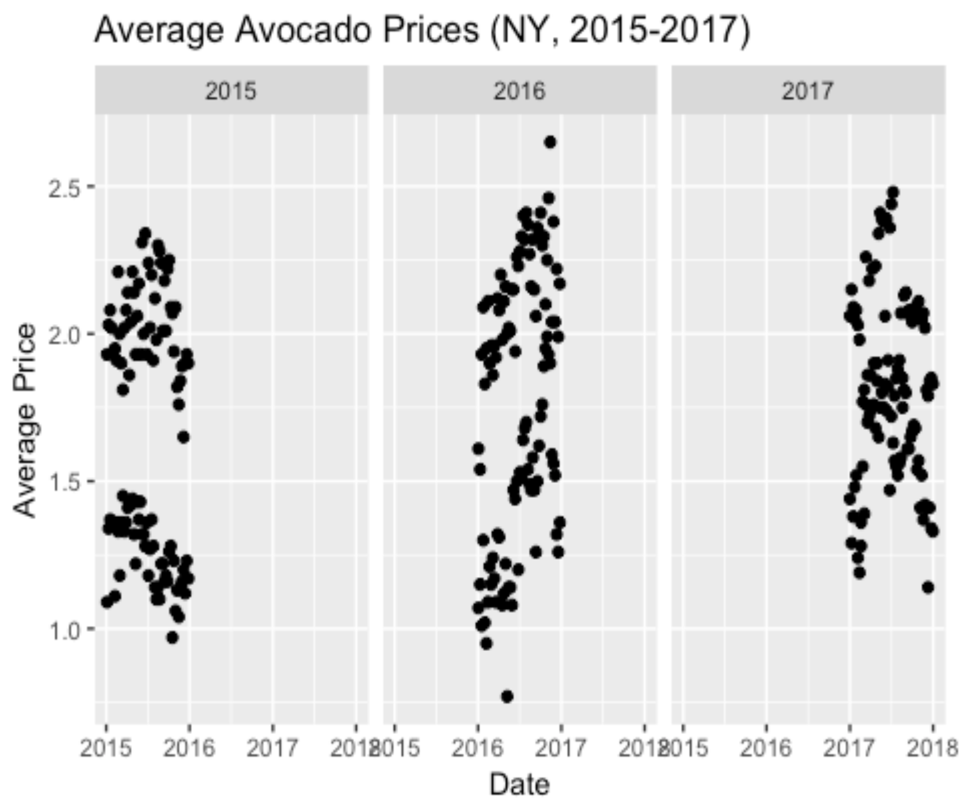
```
#filter(data frame, variable1 == "observation", variable2 != "observation")
head(NY_15_17) #preview data frame
```

```
## # A tibble: 6 x 13
##   Date       Average_Price Total_Volume `4046` `4225` `4770` Total_Bags
##   <date>             <dbl>        <dbl>  <dbl>  <dbl>  <dbl>      <dbl>
## 1 2015-01-04          1.09     1402890. 23641  1.13e6 1.87e3    249496.
## 2 2015-01-04          1.93       17328.  2357. 1.27e4 9.47e0      2269.
## 3 2015-01-11          1.34     1018226. 15881. 7.15e5 2.32e3    285499.
## 4 2015-01-11          2.03       14818.  1744. 1.09e4 6.09e1      2111.
## 5 2015-01-18          1.37     1044281. 18946. 7.49e5 3.04e3    272986.
## 6 2015-01-18          2.08        9205.   693. 7.15e3 3.79e1      1327.
## # ... with 6 more variables: Small_Bags <dbl>, Large_Bags <dbl>,
## #   XLarge_Bags <dbl>, Type <chr>, Year <dbl>, Region <chr>
```

Taking a look at this new data frame, we can see under “Region” that it only contains observations equal to “NewYork”, and that under “Year”, it does not contain any observations for “2018”. When filtering a data frame, we could also filter with other comparisons including >, >=, <, <= and logical operators like & “and”, | “or” and ! “not”.

Now that we have our data selected, let’s create a scatterplot of the average price of avocados throughout the year split by year. We can do this with a ggplot point plot and facet it by year; or in other words, create a separate plot for each year.

```
ggplot(data = NY_15_17) + geom_point(mapping = aes(x = Date, y = Average_Price)) + facet_grid(~ Year
) + ggtitle("Average Avocado Prices (NY, 2015-2017)") + labs(y = "Average Price") #create scatterplot of a
verage price in NY by date faceted by year
```



From these plots we can see that in all years, the average price of avocados fluctuated throughout the year. Avocado prices in 2015 and 2017 peaked mid-year while prices in 2016 stayed higher towards the end of the year. Additionally, 2017 prices seem to be higher than the previous two years.

Now let’s take a look at how the number of avocados sold affected the average prices. For this we will plot “Total_Volume” vs. “Average_Price”.

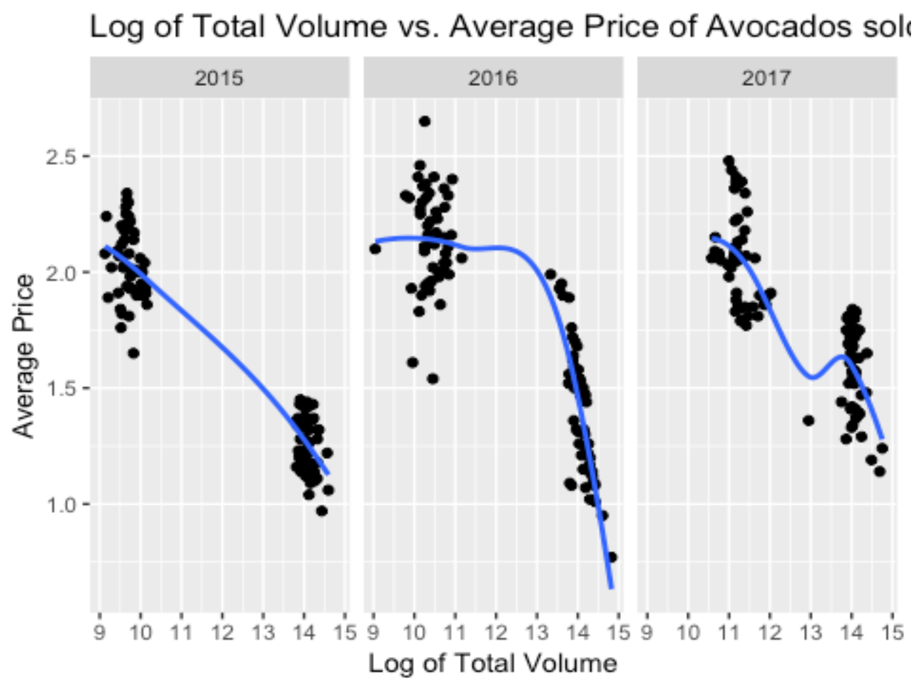
Before we plot, looking at our data frame, we can see that observations for “Total_Volume” range across several orders of magnitude which may not result in a very clean plot. We can use the handy log transformation helper function to take the log of our “Total_Volume” column in order to work with smaller values. Not only that, but we can take the log of our “Total_Volume” column and add it to our existing NY_15_17 data frame with another one of our data transformation functions: mutate(). (The log() function is just one of several operators that can be used within the mutate() function.) Let’s take a look at the log() and mutate() functions in action:

```
NY_15_17 <- mutate(NY_15_17, Total_Volume_Log = log(Total_Volume)) #create "Total_Volume_Log" v
ariable and add to data frame
#mutate(df, NewVariable = log(VarName))
colnames(NY_15_17) # print data frame variable names

##  [1] "Date"          "Average_Price"    "Total_Volume"
##  [4] "4046"          "4225"          "4770"
##  [7] "Total_Bags"      "Small_Bags"      "Large_Bags"
## [10] "XLarge_Bags"     "Type"            "Year"
## [13] "Region"          "Total_Volume_Log"
```

Taking another look at the column names of the data frame, we can see that the new column called “Total_Volume_Log” with the log values of “Total_Volume” was added to our data frame, and we were able to do that in just one step. It is also possible to add a new variable to a data frame and only keep that variable, by using the transmute() function instead. However, we still need our other variables so we will not be using that.

Now we can return to our “Total_Volume” vs. “Average_Price” plot, but this time use our new “Total_Volume_Log” variable. We can also add a smooth line over our point line so that we can clearly see the trend between the number of avocados sold and the average price.

```
ggplot(data = NY_15_17) + geom_point(mapping = aes(x = Total_Volume_Log, y = Average_Price)) + geo
m_smooth(se = FALSE, mapping = aes(x = Total_Volume_Log, y = Average_Price)) + facet_grid(~Year) + g
gtitle("Log of Total Volume vs. Average Price of Avocados sold (NY, 2015-2017)") + labs(x = "Log of Total
Volume", y = "Average Price")

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

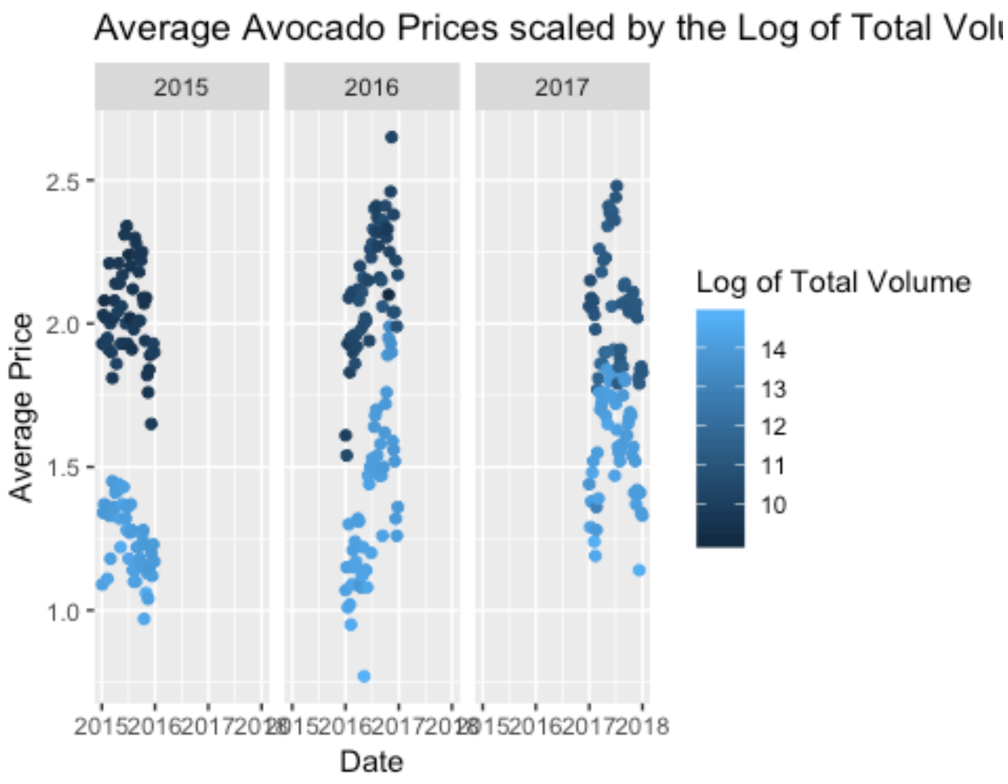


```
#create scatterplot and smooth line of total_volume_log in NY by date faceted by year
```

Do these plots remind you of something? How about the supply and demand model? We can see that when the volume of avocados sold was higher, the average price was lower.

Based on this information, if we think back to our original plot of average price throughout the year, we would expect that the average price dips correlated with larger volume of avocados sold and vice versa. By scaling our original data points with color using "Total_Volume_Log" data, we can actually see if this was true:

```
ggplot(data = NY_15_17) + geom_point(mapping = aes(x = Date, y = Average_Price, color = Total_Volum
e_Log)) + facet_grid(~ Year) + ggtitle("Average Avocado Prices scaled by the Log of Total Volume (NY, 20
15-2017)") + labs(y = "Average Price", color = "Log of Total Volume")
```



```
#create scatterplot of average price in NY by date, scaled by the log total volume, faceted by year
```

And to no surprise, the lighter blue points which represent a higher volume of avocados sold are reflected in the data points where the average price was lower. And vice versa, this correlates with higher average prices at lower volumes of avocados sold.

Now, for our final dplyr data transformation function we will focus on the "Average_Price" of avocados sold in NY. Specifically, we will find the median value of "Average_Price" by using the summarize() function. This will collapse all observations of the "Average_Price" variable into one single observation.

```
summarize(NY_15_17, Median_Avg_Price = median(Average_Price, na.rm = TRUE))

## # A tibble: 1 x 1
##   Median_Avg_Price
##              <dbl>
## 1             1.81
```

```
#summarize Average_Price with median()
#na.rm = TRUE removes any NA results
```

Using summarize(), we just collapsed the “Average_Price” variable into one row that contains its median, which shows that the median average price of avocados sold in New York from 2015-2017 was $1.81. The summarize() function is even more useful when combined with the group_by() function, which allows us to use an individual variable or group as the unit of analysis instead of the entire data frame.

For example, instead of summarizing the “Average_Price” by all variables in the data frame, we can find the median by year with the group_by() function. First, we will create a new data frame grouped by “Year”, and then summarize that data frame.

```
by_year <- group_by(NY_15_17, Year) #create data frame that groups data by Year
summarize(by_year, Median_Avg_Price = median(Average_Price, na.rm = TRUE)) #summarize Average_
Price with median() by Year

## # A tibble: 3 x 2
##    Year Median_Avg_Price
##   <dbl>            <dbl>
## 1  2015             1.55
## 2  2016             1.90
## 3  2017             1.81
```

Now, instead of collapsing the “Average_Price” variable into one row that contains the median for the entire data frame, it is now collapsed by the data frame grouped by “Year”. See the difference? In 2015, the median average price was $1.55; in 2016, the median average price was $1.90; and in 2017, the median average price was $1.81.

An easier way to accomplish the two steps above of: 1) creating the data frame grouped by “Year” and 2) summarizing the data is by using a pipe to combine the two steps into one.

By using a pipe, %>%, we can accomplish both of these steps and print the results. See the code below:

```
by_year <- NY_15_17 %>% #call the price_NY_15_17 df
  group_by(Year) %>% #group by "Year"
  summarize(Median_Avg_Price = median(Average_Price), na.rm = TRUE) %>% #summarize Average_Pri
ce with median() by Year
print(by_year) #print results

## # A tibble: 3 x 3
##    Year Median_Avg_Price na.rm
##   <dbl>            <dbl> <lgl>
## 1  2015             1.55 TRUE
## 2  2016             1.90 TRUE
## 3  2017             1.81 TRUE
```

We just combined a three-step process into one step. Imagine how useful this can be when working with larger data sets and using more steps or operations.

These five dplyr functions can do wonders when it comes to formatting data sets, and as we saw, each function has their own set of functions which makes transforming data sets even easier. We have only scraped the surface of what is possible with data transformation, and hopefully we can explore some other functions in future projects.

Data Source:

Kiggins, J. (2018, June 06). Avocado Prices. Retrieved April 27, 2019, from https://www.kaggle.com/neuromusic/avocado-prices#avocado.csv