Power of dplyr: A Perspective on Data Manipulation

Tony Hsu

10/15/2017

```
#Loading Packages
library(ggplot2)
library(png)
library(grid)
library(readr)
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
```

Introduction and Motivation

Let's suppose you are a researcher at YouTube, and you want to study which category of videos obtain the most average views. You have a data table that details videos' information such as title, category, and number of views. One way of going about this task is to group the videos with the same category and obtain each category's average view. To complete this operation, one might use the traditional R base functions to manipulate data table which typically involves bracket notation `dat[ , ]`, dollar operator `dat$name`, and the `aggregate` function.

This operation might become cumbersome and not very human friendly to read. However, the R package `dplyr` provides powerful tools for data cleaning and manipulating. With functions such as `select()`, `dplyr` makes the data analyst's job much easier. It not only simplifies the operation but it also makes it more user friendly.

For example, with the following table, which is obtained from Kaggle, aggregation would look something like this. `aggregate(youtube$views, by = list(youtube$category_id), FUN = mean)`

The data set comes from Kaggle(https://www.kaggle.com/datasnaek/youtube).

```
#Loading the data frame youtube and displaying a few rows
youtube = read_csv("../data/GBvideos.csv")
```

```
## Parsed with column specification:
## cols(
##   video_id = col_character(),
##   title = col_character(),
##   channel_title = col_character(),
##   category_id = col_integer(),
##   tags = col_character(),
##   views = col_integer(),
##   likes = col_integer(),
##   dislikes = col_integer(),
##   comment_total = col_integer(),
##   thumbnail_link = col_character(),
##   date = col_double()
## )
```

```
## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 1)
```

```
## Warning: 4 parsing failures.
## row # A tibble: 4 x 5 col     row   col               expected      actual                   file expected   <i
nt> <chr>                  <chr>       <chr>                  <chr> actual 1  2397  date no trailing characters 17
yxJDFvTRM '../data/GBvideos.csv' file 2  2397  <NA>              11 columns  21 columns '../data/GBvideos.csv' row
3  2796  date no trailing characters t2oVUxTV4WA '../data/GBvideos.csv' col 4  2796  <NA>              11 columns
21 columns '../data/GBvideos.csv'
```

```
#Showing a few rows of the data frame
head(youtube)
```

```
## # A tibble: 6 x 11
##      video_id
##         <chr>
## 1 jt2OHQh0HoQ
## 2 AqokkXoa7uE
## 3 YPVcg45W0z4
## 4 T_PuZBdT2iM
## 5 NsjsmgmbCfc
## 6 zZ2CLmvqfXg
## # ... with 10 more variables: title <chr>, channel_title <chr>,
## #   category_id <int>, tags <chr>, views <int>, likes <int>,
## #   dislikes <int>, comment_total <int>, thumbnail_link <chr>, date <dbl>
```

The purpose of this blog post is provide demonstrate why `dplyr` is a widely used package as well a discussion about its background. It has simplified a lot of data manipulation processes and provide tools for other tasks such as joining tables. Many core functions are described in https://www.r-bloggers.com/useful-dplyr-functions-wexamples/. Today, I will show these functions using a few interesting data sets.

Background

```
#Showing a logo of dplyr
img <- readPNG("../Images/dplyr.png")
grid.raster(img)
```

Here's a delightful logo of dplyr.

According to http://stat545.com/block009_dplyr-intro.html, `dplyr` is a package developed by Hadley Wickham and Romain Francois. This package is intended to be "fast, highly expressive, and open-minded about your data is stored." `dplyr` is actually tied to an earlier package, which is called `plyr`. Although `plyr` covers a wide range of inputs such as `lists` and `arrays`, `dplyr`'s main focus is on data frames. `dplyr` is actually installed as a core package of the `tidyverse` meta-package.

If you are a user of the base R functions, including functions like `apply()` and `aggregate()`, then perhaps you would find the functions within the `dplyr` to be beneficial. Moreover, if you use `for()` loops lot, this package can also save you some trouble. Perhaps you are just starting to get used to `dplyr`; we will also talk more advanced yet extremely powerful functions like `left_join`. Besides its main functions, its features such as `%>%` takes user-friendliness to the next level, as described in http://genomicsclass.github.io/book/pages/dplyr_tutorial.html.

Examples

In this section, we will go back to the first example and compute each category's average views using `dplyr`. Moreover, we will explore other funtionalities that fully demonstrates the power of `dplyr`.

`%>%`

Before we start dealing with the `dplyr` functions, I want to introduce a power tool of `dplyr` that is the pipe operator: `%>%`. Normally, you would nest functions in order perform a complex task. However, that can sometimes make it hard for other users to understand exactly what your function does. Or you would store the newly made data frame in a variable in order to perform further operation on it.

```
#Performing nested functions to select title of videos that have more than 5000 likes without Dplyr
head(select(filter(youtube, likes >
5000), title))
```

```
## # A tibble: 6 x 1
##                                                                        title
##                                                                        <chr>
## 1 Live Apple Event - Apple September Event 2017 - iPhone 8, iPhone X, iOS 11
## 2                                                 My DNA Test Results! I'm WHAT?!
## 3 getting into a conversation in a language you don't actually speak that wel
## 4                                                         Baby Name Challenge!
## 5          REVEALED - FIFA 18 stats for Chelsea's Hazard, Luiz & Christensen!
## 6                      Juicy Chicken Breast - You Suck at Cooking (episode 65)
```

This, however, does not have a high readability. `%>%` allows the user to import the output from one function to the input of another function. `youtube` becomes the first argument for `filter`. As shown in http://genomicsclass.github.io/book/pages/dplyr_tutorial.html, it is extremely powerful in cases of multi-levels operations.

```
#Performing nested functions to select title of videos that have more than 5000 likes with Dplyr
youtube %>%
  filter(likes > 5000) %>%
  select(title) %>%
  head()
```

```
## # A tibble: 6 x 1
##                                                                          title
##                                                                          <chr>
## 1 Live Apple Event - Apple September Event 2017 - iPhone 8, iPhone X, iOS 11
## 2                                         My DNA Test Results! I'm WHAT?!
## 3 getting into a conversation in a language you don't actually speak that wel
## 4                                                     Baby Name Challenge!
## 5          REVEALED - FIFA 18 stats for Chelsea's Hazard, Luiz & Christensen!
## 6                     Juicy Chicken Breast - You Suck at Cooking (episode 65)
```

As you can see the code is much cleaner than the first one.

`groupby()` and `summarise()`

Here, I will introduce some `dplyr` functions. The `group_by()` function puts the data frame in different parts according to some variable. A function that is often used together with `group_by()` is the `summarise()` function. `summarise()` function takes a data frame and computes whichever statistics you want: `sd()`, `max()`, or `median()`.

Now, let's tackle the main question.

```
#Grouping the youtube videos by categories
grouped_categories = youtube %>%
  group_by(categories = category_id) %>%
  summarise(avg_mean_by_cat = mean(views))
grouped_categories
```

```
## # A tibble: 15 x 2
##    categories avg_mean_by_cat
##         <int>           <dbl>
##  1          1      1110808.99
##  2          2      1877787.98
##  3         10      2239880.70
##  4         15      1086241.69
##  5         17       603660.96
##  6         19       188051.47
##  7         20       544966.19
##  8         22       778975.98
##  9         23      1527209.13
## 10         24      1193536.55
## 11         25       871629.28
## 12         26       428835.75
## 13         27       557994.15
## 14         28      1025574.45
## 15         29        45752.75
```

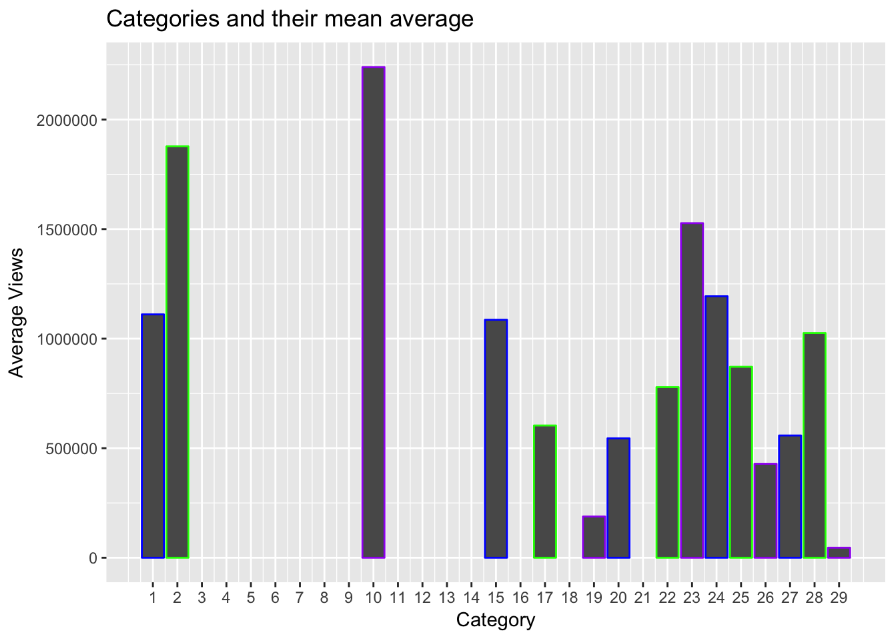
We can verify our result with the other approach using base R functions.

```
#Using the base R functions to group categories
aggregate(youtube$views,
by = list(youtube$category_id),
FUN = mean)
```

```
##    Group.1          x
## 1        1 1110808.99
## 2        2 1877787.98
## 3       10 2239880.70
## 4       15 1086241.69
## 5       17  603660.96
## 6       19  188051.47
## 7       20  544966.19
## 8       22  778975.98
## 9       23 1527209.13
## 10      24 1193536.55
## 11      25  871629.28
## 12      26  428835.75
## 13      27  557994.15
## 14      28 1025574.45
## 15      29   45752.75
```

Let's visualize our result.

```r
#Using ggplot to plot a barchart of the grouped categories.
ggplot(grouped_categories, aes(x = categories,
y = avg_mean_by_cat)) + geom_bar(stat = "identity",
color = rep(c("blue", "green", "purple"), 5)) +
ggtitle("Categories and their mean average") +
xlab("Category") +
ylab("Average Views") +
scale_x_continuous(breaks = round(seq(
min(grouped_categories$categories),
max(grouped_categories$categories),
by = 1
), 1))
```



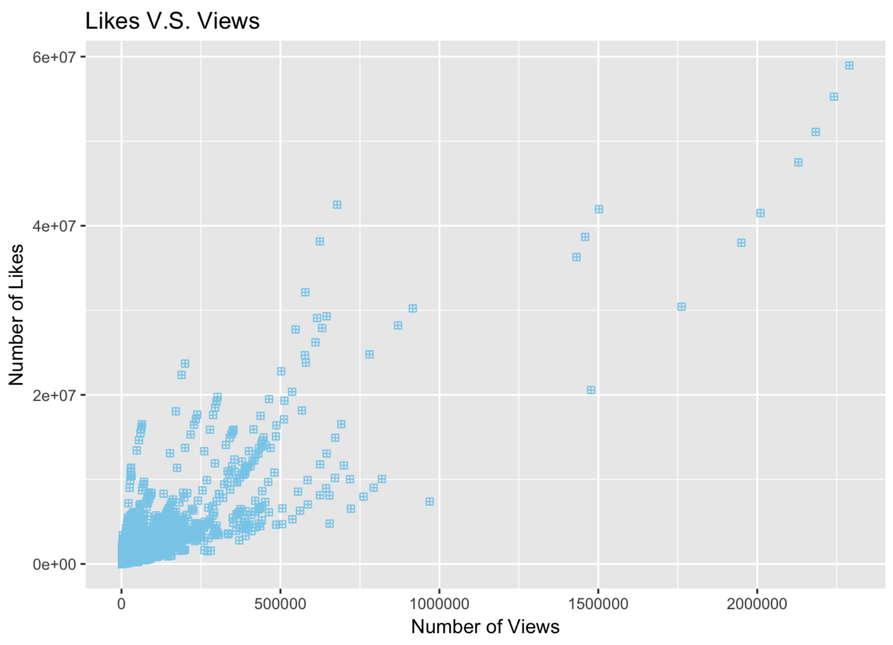
Using the chart, we can tell that category 10 is has the highest average views. The second highest average views is category 2. On the other hand, the lowest average views is category 29. Perhaps it is because our data set doesn't have data about some groups, but we coud not display average views for some groups.

`select()` and `filter()`

The function `select` takes certain columns and form a new data frame. The function `filter()` takes the rows that meet the criteria.

Let's suppose that you want to examine if there's any correlation between the number of likes and the nubmber of views. Furthermore, we wish to select the videos that have more than 1000 likes.

```r
#Using select and filter to make a data frame of columns likes and views that only have more than 1000 likes.
likes_views = youtube %>%
  filter(likes > 1000) %>%
  select(likes, views)

#Using ggplot to plot a scatterplot likes and views.
ggplot(likes_views) + geom_point(aes(x = likes,
y = views), shape = 12, color = "skyblue") +
  ggtitle("Likes V.S. Views") +
  xlab("Number of Views") +
  ylab("Number of Likes")
```



From this chart, we can see that there seems to be a positive correlation between the number of the views and the number of likes for a video. Although a large of group of videos are below 500,000 views, they are videos above 500,0000 views. There are videos as far as above 2,000,000 views.

`join()`

Another powerful usage of `dplyr` is joining tables. `dplyr` offers various ways for the user to join tables: `left_join(table_1, table_2, by = "Player")`, `right_join(table_1, table_2, by = "Player")`, `inner_join(table_1, table_2, by = "Player")`, `full_join(table_1, table_2, by = "Player")`. Left join would join the rows from table 2 to table 1, retaining any row of table 2 even if there's no match. Right join is the same process except it would join take rows from table 1 and join them with table 2. Inner join only joins tables with the mathced column. Full join retains rows even if there's no match.

Let's suppose that you have a data frame of trainers and their list of pokemons and a data frame of pokemons and their statistics. You would like to combine the tables in order to get statistics of the trainer's specific pokemons. This data set comes from https://www.kaggle.com/abcsds/pokemon .

```
#Loading the data frames: Pokemon and trainers
Pokemon = read_csv("../data/Pokemon.csv")
```

```
## Parsed with column specification:
## cols(
##   `#` = col_integer(),
##   Name = col_character(),
##   `Type 1` = col_character(),
##   `Type 2` = col_character(),
##   Total = col_integer(),
##   HP = col_integer(),
##   Attack = col_integer(),
##   Defense = col_integer(),
##   `Sp. Atk` = col_integer(),
##   `Sp. Def` = col_integer(),
##   Speed = col_integer(),
##   Generation = col_integer(),
##   Legendary = col_character()
## )
```

```
trainers = read_csv("../data/trainers.csv")
```

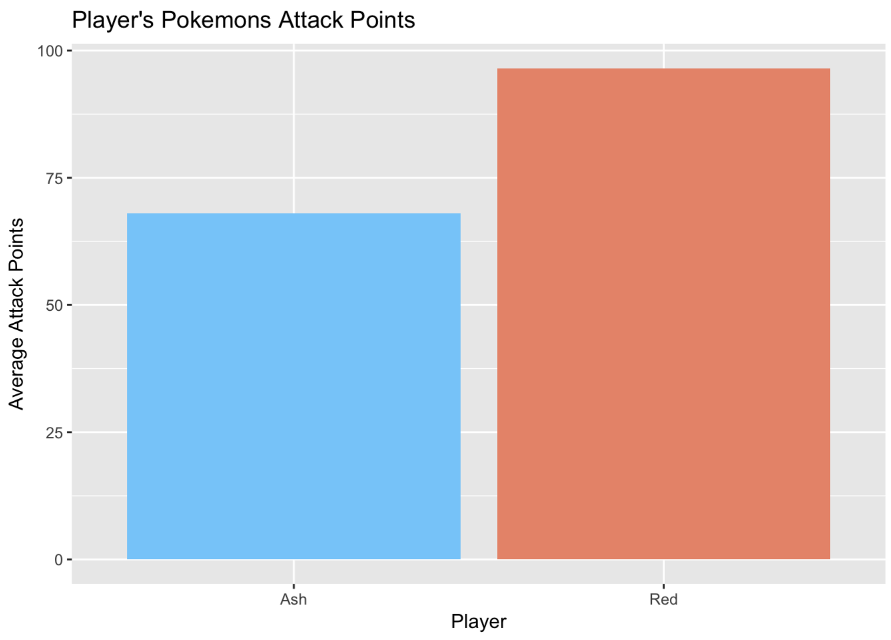
```
## Parsed with column specification:
## cols(
##   player = col_character(),
##   `Pokemon Name` = col_character()
## )
```

```
#Renaming the Name column of Pokemon to Pokemon Name so that we can join tables
Pokemon = rename(Pokemon, `Pokemon Name` = Name)

#Joining the tables and displaying the join tabled
Trainers_pokemons  = inner_join(trainers, Pokemon, by = "Pokemon Name")
head(Trainers_pokemons, 10)
```

```
## # A tibble: 5 x 14
##   player `Pokemon Name`   `#` `Type 1` `Type 2` Total    HP Attack Defense
##    <chr>          <chr> <int>    <chr>    <chr> <int> <int>  <int>   <int>
## 1    Ash      Charizard     6     Fire   Flying   534    78     84      78
## 2    Ash        Pikachu    25 Electric     <NA>   320    35     55      40
## 3    Ash       Venomoth    49      Bug   Poison   450    70     65      60
## 4    Red        Snorlax   143   Normal     <NA>   540   160    110      65
## 5    Red      Blastoise     9    Water     <NA>   530    79     83     100
## # ... with 5 more variables: `Sp. Atk` <int>, `Sp. Def` <int>,
## #   Speed <int>, Generation <int>, Legendary <chr>
```

Let’s visualize the attack points using Ash’s pokemons v.s. Red’s pokemons.

```
#Grouping the pokemons according to their trainers
grouped_trainers = Trainers_pokemons %>%
  group_by(player) %>%
  summarise(avg_attack = mean(Attack))

#Barplotting the attack points of pokemons according to their owners.
  ggplot(grouped_trainers,
  aes(x = player , y = avg_attack)) +
  geom_bar(stat = "identity", fill =
  rep(c("lightskyblue", "darksalmon"))) +
  ggtitle("Player's Pokemons Attack Points") +
  xlab("Player") +
  ylab("Average Attack Points")
```



Here we can see that the average attack points of Red’s pokemons is greater than the average attack points of Ash’s pokemons. The average attack points of Ash’s pokemons is below 75. In contrast, the average attack points of Red’s pokemons is close to 100.

If you want retain the pokemons who don’t have a trainer or the trainer is not in the table. You can use `right_join`.

```
rightJoinTable = right_join(trainers, Pokemon, by = "Pokemon Name")
head(rightJoinTable, 10)
```

```
## # A tibble: 10 x 14
##    player            `Pokemon Name`   `#` `Type 1` `Type 2` Total    HP
##     <chr>                     <chr> <int>    <chr>    <chr> <int> <int>
##  1   <NA>                 Bulbasaur     1    Grass   Poison   318    45
##  2   <NA>                   Ivysaur     2    Grass   Poison   405    60
##  3   <NA>                  Venusaur     3    Grass   Poison   525    80
##  4   <NA>     VenusaurMega Venusaur     3    Grass   Poison   625    80
##  5   <NA>                Charmander     4     Fire     <NA>   309    39
##  6   <NA>                Charmeleon     5     Fire     <NA>   405    58
##  7    Ash                 Charizard     6     Fire   Flying   534    78
##  8   <NA> CharizardMega Charizard X     6     Fire   Dragon   634    78
##  9   <NA> CharizardMega Charizard Y     6     Fire   Flying   634    78
## 10   <NA>                  Squirtle     7    Water     <NA>   314    44
## # ... with 7 more variables: Attack <int>, Defense <int>, `Sp. Atk` <int>,
## #   `Sp. Def` <int>, Speed <int>, Generation <int>, Legendary <chr>
```

Here, if there’s no match of trainer, then the trainer would be N/A.

Discussion and Conclusions

After utilizing the `dplyr` throughout homeworks and labs, I agreed with stat545.com that is very fast and highly expressive. The package takes away the cumbersomeness of the base functions. Moreover, with its joining tables functions resembles the functionality of SQL. Right, left, and full join have the power to retain rows even if there's no match. As data analysis integreates more into your life, `dplyr` 's powerfulness would certainly benefit you.

The appliation of `dplyr` goes beyond data manipulation. In many industries, it is widely as a tool for feature engineering and data wrangling. This video (https://www.youtube.com/watch?v=Ds6arVTWwDc) explores the `dplyr` with the Kaggle titanic dataset!

Here's another data analysis project using `dplyr` (https://rstudio-pubs-static.s3.amazonaws.com/215885_e3c935b8b5524576bec6f3ddb48e0458.html). This project deals with loans. As you can see, the user can easily add in new columns with a good readability.

Thank you for reading for my post about `dplyr`. I hope after this post, you're more comfortable with data manipulation.

References

- https://www.kaggle.com/datasnaek/youtube
- https://www.kaggle.com/abcsds/pokemon
- https://www.r-bloggers.com/useful-dplyr-functions-wexamples/
- http://stat545.com/block009_dplyr-intro.html
- http://genomicsclass.github.io/book/pages/dplyr_tutorial.html
- https://www.youtube.com/watch?v=Ds6arVTWwDc
- https://rstudio-pubs-static.s3.amazonaws.com/215885_e3c935b8b5524576bec6f3ddb48e0458.html